2731
2664

#4

PTO/SB/21 (08-00)
Approved for use through 10/31/2002. OMB 0651-0031
U.S. Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE

TRANSMITTAL FORM

(to be used for all correspondence after initial filing)

Application Number	09/602,418
Filing Date	06/23/2000
First Named Inventor	MAREK MUSIAL
Group Art Unit	2731
Examiner Name	
Total Number of Pages in This Submission	190
Attorney Docket Number	6963 US

ENCLOSURES *(check all that apply)*

- [ ] Fee Transmittal Form
  - [ ] Fee Attached
- [ ] Amendment / Reply
  - [ ] After Final
  - [ ] Affidavits/declaration(s)
- [ ] Extension of Time Request
- [ ] Express Abandonment Request
- [ ] Information Disclosure Statement
- [x] Certified Copy of Priority Document(s)
- [ ] Response to Missing Parts/ Incomplete Application
  - [ ] Response to Missing Parts under 37 CFR 1.52 or 1.53
- [ ] Assignment Papers *(for an Application)*
- [ ] Drawing(s)
- [ ] Licensing-related Papers
- [ ] Petition
- [ ] Petition to Convert to a Provisional Application
- [ ] Power of Attorney, Revocation Change of Correspondence Address
- [ ] Terminal Disclaimer
- [ ] Request for Refund
- [ ] CD, Number of CD(s) _____
- [ ] After Allowance Communication to Group
- [ ] Appeal Communication to Board of Appeals and Interferences
- [ ] Appeal Communication to Group *(Appeal Notice, Brief, Reply Brief)*
- [ ] Proprietary Information
- [ ] Status Letter
- [x] Other Enclosure(s) *(please identify below)*: Return Post Card

Remarks

German Priority Document No. 199 29 166.7

SIGNATURE OF APPLICANT, ATTORNEY, OR AGENT

Firm or Individual name	Francis I. Gray, TEKTRONIX, INC.
Signature	Francis I. Gray
Date	05/13/2002

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to: Commissioner for Patents, Washington, DC 20231 on this date: 05/13/2002

Typed or printed name	Pauline Bradley		
Signature	Pauline Bradley	Date	05/13/2002

BUNDESREPUBLIK DEUTSCHLAND

Prioritätsbescheinigung über die Einreichung einer Patentanmeldung

Aktenzeichen:	199 29 166.7
Anmeldetag:	25. Juni 1999
Anmelder/Inhaber:	Tektronix, Inc., Wilsonville, Oreg./US
Bezeichnung:	Verfahren zum Erlernen von Protokollregeln aus beobachteten Kommunikationsabläufen
IPC:	G 06 F, H 04 L

Die angehefteten Stücke sind eine richtige und genaue Wiedergabe der ursprünglichen Unterlagen dieser Patentanmeldung.

München, den 04. April 2002
Deutsches Patent- und Markenamt
Der Präsident
Im Auftrag

Agurks

A 9161
06/00
EDV-L

STRASSE & HOFSTETTER

PATENTANWÄLTE
EUROPEAN PATENT ATTORNEYS
EUROPEAN TRADEMARK ATTORNEYS

München
Dr.rer.nat. ALFONS J. HOFSTETTER [1,2]
Dr.-Ing. EDUARD F. SCHURACK [1,2]

Hannover
Dipl.-Ing. MICHAEL SKORA [1,2]

Dipl.-Ing. JOACHIM STRASSE (1962-1998) [1,2,3]
Dr. jur. PETER C. BITTNER [4]

[1] Patentanwalt
[2] Zugelassener Vertreter beim Europäischen Patentamt und Europäischen Harmonisierungsamt für den Binnenmarkt
[3] Consultant / Nur in beratender Tätigkeit s. 1999
[4] Consultant / Nur in beratender Tätigkeit s. 1997

In Zusammenarbeit mit / In cooperation with

LENZING GERBER
Patentanwälte · Düsseldorf

Anwaltsakte: 25146

Tektronix, Inc.
26600 S.W. Parkway Avenue
P.O. Box 1000
Wilsonville, Oregon 97070-1000
U.S.A.

DEUTSCHE PATENTNEUANMELDUNG

Verfahren zum Erlernen von Protokollregeln aus beobachteten Kommunikationsabläufen

---

BALANSTRASSE 57 TELEFON +49/(0)89 - 4509180
D-81541 MÜNCHEN FAX +49/(0)89 - 45091812
TIERGARTENSTRASSE 122 TELEFON +49/(0)511 - 5106365
D-30559 HANNOVER Fax +49/(0)511 - 5106371

EMAIL Strasse-Hofstetter@T-Online.de

Inhaltsverzeichnis

Tabellenverzeichnis

Tektronix, Inc.,
Wilsonville, Oregon 97070, U.S.A.
Deutsche Patentneuanmeldung

Anwaltsakte: 25146

Verfahren zum Erlernen von Protokollregeln aus beobachteten Kommunikationsabläufen

---

Patentansprüche

1. Verfahren zum Erlernen des einer Protokollimplementierung zugrundeliegenden endlichen Automaten,
dadurch gekennzeichnet,
daß es die Zeitpunkte innerhalb einer Beispielkommunikation in Äquivalenzklassen zusammenfaßt und diese Äquivalenzklassen als Zustände des erlernten Automaten verwendet.

2. Verfahren nach Anspruch 1,
dadurch gekennzeichnet,
daß es zur Bildung der Äquivalenzklassen ein Ähnlichkeitsmaß zwischen jeweils zwei Zeitpunkten innerhalb der Beispielkommunikation berechnet, das von der Länge der für beide Zeitpunkte übereinstimmenden, die Zeitpunkte umgebenden PDU-Sequenz abhängt.

3. Verfahren nach Anspruch 1 oder 2,
dadurch gekennzeichnet,
daß es durch eine untere Schranke für den Wert eines Ähnlichkeitsmaßes zwischen jeweils zwei Zeitpunkten innerhalb der Beispielkommunikation eine Ähnlichkeitsrelation so definiert, daß zwei Zeitpunkte die Ähnlichkeitsrelation erfüllen, wenn das Ähnlichkeitsmaß zwischen diesen beiden Zeitpunkten größer oder gleich der unteren Schranke ist.

4. Verfahren nach einem der vorhergehenden Ansprüche, dadurch gekennzeichnet,
daß es die die Äquivalenzklassen bildende Äquivalenzrelation berechnet, indem es die transitive Hülle einer Ähnlichkeitsrelation zwischen den Zeitpunkten innerhalb der Beispielkommunikation bildet.

5. Verfahren nach einem der vorhergehenden Ansprüche, dadurch gekennzeichnet,
daß es alle PDUs der Beispielkommunikation als Zustandsübergänge des erlernten Automaten einträgt, und zwar jeweils als Übergang von demjenigen Zustand, in dessen Äquivalenzklasse der Zeitpunkt unmittelbar vor der betreffenden PDU liegt, zu demjenigen Zustand, in dessen Äquivalenzklasse der Zeitpunkt unmittelbar nach der betreffenden PDU liegt, markiert mit der betreffenden PDU.

6. Verfahren nach einem der vorhergehenden Ansprüche, dadurch gekennzeichnet,
daß es die Verfahrensschritte nach mindestens einem der Ansprüche 1 bis 5, vorzugsweise nach der Kombination der Ansprüche 1 bis 5, mehrfach für überlappende Teilbereiche der Beispielkommunikation durchführen kann, wobei die Ähnlichkeitsrelationen jeweils zweier überlappender Teilbereiche zu einer gemeinsamen Äquivalenzrelation vereinigt werden.

7. Verfahren, das einen möglichen zugrundeliegenden endlichen Automaten einer Protokollimplementierung anhand einer Beispielkommunikation erlernt, wobei der zugrundeliegende endliche Automat aus den Grundzuständen des Protokolls besteht und seine Zustandsübergänge mit den PDU-Typen des Protokolls markiert sind,
dadurch gekennzeichnet, daß es

a) alle Zeitpunkte in der Beispielkommunikation zwischen zwei jeweils aufeinanderfolgenden PDUs betrachtet;

b) für jedes Paar solcher Zeitpunkte nach a) ein Ähnlichkeitsmaß berechnet, das die Summe der Anzahl der für beide Zeitpunkte übereinstimmenden PDU-Typen vor dem Zeitpunkt und der Anzahl der für beide Zeitpunkte übereinstimmenden PDU-Typen nach dem Zeitpunkt angibt;

c) durch eine untere Schranke für den Wert des nach b) berechneten Ähnlichkeitsmasses eine Ähnlichkeitsrelation zwischen jeweils zwei Zeitpunkten definiert, derart, daß zwei Zeitpunkte die Ähnlichkeitsrelation erfüllen, wenn das Ähnlichkeitsmaß zwischen diesen beiden Zeitpunkten größer oder gleich der unteren Schranke ist;

d) die transitive Hülle für die nach c) definierte Ähnlichkeitsrelation bildet, wobei die transitive Hülle eine Äquivalenzrelation auf Zeitpunkten nach a) darstellt;

e) jede Äquivalenzklasse der Äquivalenzrelation nach d) als einen Zustand des erlernten Automaten verwendet;

f) alle PDUs der Beispielkommunikation als Zustandsübergänge des erlernten Automaten einträgt, und zwar als Übergang von demjenigen Zustand, in dessen Äquivalenzklasse gemäß e) der Zeitpunkt unmittelbar vor der betreffenden PDU liegt, zu demjenigen Zustand, in dessen Äquivalenzklasse gemäß e) der Zeitpunkt unmittelbar nach der betreffenden PDU liegt, markiert mit dem PDU-Typ der betreffenden PDU, wobei Übergänge, die in Ausgangs- und Folgezustand und im PDU-Typ übereinstimmen, nur einmal eingetragen werden;

g) zum Zwecke der Rechenzeit- und Speicherbedarfsbegrenzung den Ablauf nach a) bis f) mehrfach für überlappende Teilbereiche der Beispielkommunikation

durchführen kann, wobei die Ähnlichkeitsrelationen gemäß c) jeweils zweier überlappender Teilbereiche vereinigt werden und die zustandsbildende Äquivalenzrelation analog zu d) über die Vereinigung der Ähnlichkeitsrelationen berechnet wird.

8. Verfahren zum Erlernen arithmetischer Klassifikationsregeln für Merkmalsvektoren aus einer Trainingsmenge mit positiven Beispielen,
dadurch gekennzeichnet,
daß es anhand statistischer Maße abgeleitete Merkmale in Form arithmetischer Terme bildet und logische Bedingungen auf den Zahlenwerten der Merkmale aus der Trainingsmenge oder der abgeleiteten Merkmale formuliert.

9. Verfahren nach Anspruch 8,
dadurch gekennzeichnet,
daß es eingesetzt wird, um anhand einer Beispielkommunikation einer Protokollmaschine Regeln für die numerischen PDU-Feld-Inhalte einer Sequenz von PDUs zu erlernen.

10. Verfahren nach Anspruch 8 oder 9,
dadurch gekennzeichnet,
daß es anhand der Korrelations- und Regressionskoeffizienten auf der Trainingsmenge für jedes mögliche Merkmalspaar abgeleitete Merkmale bildet, wobei der Wert eines abgeleiteten Merkmals für jedes Trainingsbeispiel berechnet wird als Summe, Produkt, Quotient oder Differenz zweier schon vorhandener Merkmale oder als Produkt aus einem vorhandenen Merkmal und einer Konstanten.

11. Verfahren nach einem der Ansprüche 8 bis 10,
dadurch gekennzeichnet,
daß es zur Bildung der Bedingungen auffällige Häufungen der Werte eines in der Trainingsmenge vorhandenen oder

abgeleiteten Merkmals bei einem Zahlenwert oder innerhalb eines Zahlenintervalls feststellt.

12. Verfahren nach Anspruch 11
dadurch gekennzeichnet,
daß eine auffällige Häufung dadurch definiert ist, daß sie den Quotienten maximiert zwischen der Breite der kleineren der beiden unmittelbar an das Zahlenintervall angrenzenden Lücken, in denen keine Werte des betreffenden Merkmals auftreten, und der Breite der größten Lücke innerhalb des Zahlenintervalls, in der keine Werte des betreffenden Merkmals auftreten.

13. Verfahren nach einem der Ansprüche 8 bis 12,
dadurch gekennzeichnet,
daß es mehrere Unterklassen der Trainingsmenge konstruiert, indem es die logischen Bedingungen in einer Disjunktion von Klauseln organisiert, wobei eine Klausel eine Konjunktion einer oder mehrerer logischer Bedingungen darstellt und jeweils eine Unterklasse der Trainingsmenge beschreibt.

14. Verfahren nach Anspruch 13,
dadurch gekennzeichnet,
daß es zur Charakterisierung der gesamten Trainingsmenge eine Auswahl der konstruierten Klauseln durchführt, so daß möglichst alle Elemente der Trainingsmenge durch mindestens eine der Klauseln ausgewählt werden und möglichst viele durch genau eine Klausel.

15. Verfahren zum Erlernen arithmetischer Klassifikationsregeln für Merkmalsvektoren aus einer Trainingsmenge mit ausschließlich positiven Beispielen, eingesetzt, um Regeln für die numerischen PDU-Feld-Inhalte einer Sequenz von PDUs zu erlernen, die einem spezifischen Teilpfad in

einem bekannten endlichen Automaten von Protokoll-Grundzuständen und PDU-Typen entspricht,
dadurch gekennzeichnnet, daß es

a) jede Komponente eines Merkmalsvektors als Ausprägung eines Attributs auffaßt, wobei die Anzahl der am Anfang vorhandenen Attribute der Dimension der Merkmalsvektoren entspricht;

b) anhand der Korrelations- und Regressionskoeffizienten auf der Trainingsmenge für jedes mögliche Attributpaar neue, sogenannte abgeleitete Attribute bildet, wobei der Wert eines abgeleiteten Attributs für jeden Merkmalsvektor aus schon vorhandenen Attributwerten dieses Vektors berechnet werden, und zwar als Summe, Produkt, Quotient oder Differenz zweier schon vorhandener Attribute oder als Produkt aus einem vorhandenen Attribut und einer Konstanten, wobei bei der Berechnung aus zwei vorhandenen Attributen die in die Berechnung eingehenden Attribute nach maximaler Korrelation zu einem dritten Attribut und die arithmetische Verknüpfung nach der maximalen Korrelation des abgeleiteten Attributs zu demselben dritten Attribut ausgewählt werden, und wobei bei der Multiplikation mit einer Konstanten ein Attribut mit einer besonders großen Korrelation zu einem zweiten Attribut mit dem linearen Regressionskoeffizienten des Attributpaares derart multipliziert wird, daß das resultierende abgeleitete Attribut dem genannten zweiten Attribut numerisch möglichst entspricht;

c) auffällige Häufungen der Werte eines ursprünglichen oder gemäß b) abgeleiteten Attributs bei einem Zahlenwert oder innerhalb eines Zahlenintervalls feststellt, wobei eine auffällige Häufung dadurch definiert ist, daß sie den Quotienten maximiert zwischen der Breite der kleineren der beiden unmittelbar an das Zahlenintervall angrenzenden Lücken, in den keine Werte des betreffenden Attributs auftre-

ten, und der Breite der größten Lücke innerhalb des Zahlenintervalls, in der keine Werte des betreffenden Attributs auftreten;

d) anhand auffälliger Häufungen gemäß c) Klauseln bildet, die jeweils eine logische Bedingung zur Auswahl derjenigen Beispiele aus der Trainingsmenge formulieren, deren Attributwerte eines bestimmten Attributs innerhalb eines nach der Charakteristik der zugehörigen auffälligen Häufung gemäß c) bestimmten Zahlenintervalls liegen, wobei jede Klausel eine Konjunktion mehrerer solcher Auswahlbedingungen für verschiedene Attribute darstellen kann;

e) zur Charakterisierung der gesamten Trainingsmenge eine Auswahl der gemäß d) konstruierten Klauseln durchführt, so daß möglichst alle Elemente der Trainingsmenge durch mindestens eine der Klauseln ausgewählt werden und möglichst viele durch genau eine Klausel.

Tektronix, Inc.,
Wilsonville, Oregon 97070, U.S.A.
Deutsche Patentneuanmeldung

Anwaltsakte: 25146

Verfahren zum Erlernen von Protokollregeln aus beobachteten Kommunikationsabläufen

---

Zusammenfassung:

Die Analyse protokollbasierter Datenkommunikation durch passive Analysatoren beschränkt sich heute meist auf die Prüfung der Nachrichtenkodierung, eine Validierung der Protokollprozeduren ist aktiven Testgeräten vorbehalten. In dieser Arbeit werden zwei neue Verfahren entwickelt, um korrektes Protokollverhalten allein durch passive Beobachtung, auch in Echtzeit, überprüfen zu können. Das erste Verfahren basiert auf einer exakten Spezifikation des beobachtbaren Sollverhaltens und verwendet unsicherheitsbehaftete Zustandsbeschreibungen, um sich auf einen laufenden Kommunikationsvorgang aufsynchronisieren zu können. Vollständigkeit und Korrektheit des Analyseverfahrens als wesentliche Qualitätskriterien werden nachgewiesen. Es kann die manuelle Sucharbeit nach Fehlerquellen um Größenordnungen reduzieren. Das zweite Verfahren umgeht den Bedarf nach einer Beschreibung des Sollverhaltens, indem es die Regeln eines Protokolls aus einer fehlerfreien Beispielkommunikation erlernt und auf zu analysierende Beobachtungen anwendet. Dieses Verfahren basiert auf zwei neuen Lernalgorithmen, einem zum effizienten Lernen optimaler endlicher Automaten und einem zur Ableitung arithmetischer Klassifikationsregeln, jeweils aus einer Stichprobe rein positiver Beispiele. Es kann zu minimalen Kosten auf neue Protokolle angewandt werden. Beide Verfahren sind als Prototypen implementiert und werden abschließend vergleichend bewertet.

Kapitel 1

Einleitung und Überblick

Im Laufe der Weiterentwicklung der elektronischen Datenverarbeitung und Telekommunikation hat der automatisierte Austausch von digitalen Informationen zwischen verschiedenen, gegeneinander abgegrenzten Einheiten ständig an Bedeutung gewonnen. Dieser Prozeß verlief und verläuft mit Sicherheit nicht linear in der Zeit, sondern weist eine immer stärkere Beschleunigung auf: Der Beginn des Computer-Zeitalters war gekennzeichnet von wenigen einzelnen Rechnersystemen, die vom jeweiligen Kunden meist für eine sehr spezielle Aufgabe eingesetzt wurden. Ein Austausch von Daten erfolgte typischerweise nur innerhalb einer solchen Rechenanlage, sofern Peripherieeinheiten wie Drucker, Lochkartengeräte und später Massenspeicher dazugerechnet werden. Bereits mit der Einführung von Mehrbenutzersystemen mit Terminals verkomplizierte sich der Datenaustausch, mit der Vernetzung einiger Großrechner setzte sich diese Entwicklung fort. Erst seit Anfang der achtziger Jahre zeichnen sich drei Entwicklungen ab, die den digitalen Informationsaustausch nach Menge und Komplexität zu potenzieren begannen:

1. Der *persönliche Computer* am Arbeitsplatz und später zu Hause. Damit vervielfacht sich die Anzahl potentieller Kommunikationsendpunkte.

2. Die Idee des *offenen Systems*, das in der Lage ist, mit anderen Systemen zu kommunizieren, deren Identität nicht a priori bekannt ist.

3. Die Digitalisierung des Fernsprechnetzes und weiterer, früher analog arbeitender Kommunikationsdienste, die nun auch jedes Telefon und jedes Rundfunkgerät zu potentiellen Teilnehmern weltweiter digitaler Kommunikation machen.

Mit der Vervielfachung der Anwendungsgebiete der digitalen Kommunikation einher geht die Einführung immer höherer Übertragungsraten in den Kommunikationstrecken. Aus 75 bit/s der ersten Fernschreiber sind Mega-, Giga- und bald auch Terabit geworden.

Voraussetzung für das einwandfreie Funktionieren digitaler Kommunikation ist, daß sich alle beteiligten Kommunikationspartner gemäß gemeinsamer Regeln verhalten. Solche Regelwerke für Kommunikationsverhalten werden *Protokolle* genannt.

Aufgrund der Offenheit, Diversität, Komplexität und Bandbreite moderner Kommunikationssysteme kommt der Entwicklung, Normung und Beschreibung von Protokollen eine ständig

größere Bedeutung zu. Im selben Maße vermehren sich die Fehlerquellen, die zu mehr oder weniger großen Störungen in Kommunikationssystemen führen. Zu typischen Fehlerquellen, die sich in modernen Breitbandkommunikationssystemen immer wieder zeigen, gehören:

- Fehlfunktionen aufgrund des Einsatzes von Protokollen oder Protokollvarianten, die im gegebenen Kontext einer Systeminstallation nicht benutzt werden dürften.
- Fehlfunktionen aufgrund von konfigurierbaren *Protokoll-Parametern*, die für den gegebenen Kontext einer Systeminstallation inkompatibel eingestellt wurden.
- Fehlfunktionen aufgrund von Überlastsituationen, in denen einzelne Komponenten des Kommunikationssystems gezwungen sind, Daten zu verwerfen.

Die Erfahrung zeigt, daß alle Bemühungen, die einwandfreie Funktion kommunikationsrelevanter Hard- und Software vor der Markteinführung durch geeignete Entwicklungsverfahren und aktive Testprozeduren sicherzustellen, nicht in jedem Fall Erfolg haben. Daraus entsteht der Bedarf nach Hilfsmitteln zur Fehlersuche, die *im laufenden Betrieb* eines Kommunikationssystems eingesetzt werden und Fehlfunktionen erkennen, einer Systemkomponente zuordnen und Anhaltspunkte zu ihrer Beseitigung liefern können.

Verfügbare Werzeuge für Messungen im laufenden Betrieb, meist als Protokoll- oder Netzwerkmonitore bezeichnet, haben gemeinsam, daß sie nicht das vollständige Kommunikationsverhalten gegen das Sollverhalten validieren können, sondern ein erhebliches Maß an manueller Prüfarbeit erfordern. Hier besteht also ein erhebliches Potential, eine sehr unangenehme manuelle Suchtätigkeit zu automatisieren.

Im Zuge einer automatisierten Verhaltensanalyse ergeben sich zwei Teilprobleme:

1. Wie muß ein Verfahren aussehen, das in der Lage ist, alle Aspekte eines Kommunikationsverhaltens effizient – möglichst in Echtzeit – zu überprüfen, indem der über eine in Betrieb befindliche Kommunikationsverbindung laufende Verkehr passiv beobachtet wird?

2. Wie kann das Sollverhalten – das ja als Grundlage der Überprüfung dient – möglichst schnell und kostengünstig dem Analysegerät zur Verfügung gestellt werden? Ist dies auch möglich, wenn das zugrundeliegende Protokoll nicht in Form einer formalen, maschinentauglichen Spezifikation vorliegt?

In dieser Arbeit wird für jedes der beiden Teilprobleme eine neue Lösung entwickelt. Während die Lösung zu Teil 1 auf einer *vorgegebenen* Spezifikation des Sollverhaltens aufbaut, soll für Teil 2 der Versuch unternommen werden, das Sollverhalten anhand von beispielhaften Kommunikationsabläufen zu *erlernen*. Dazu muß vorausgesetzt werden, daß die Beispielabläufe fehlerfrei sind, denn die Kenntnis von Fehlern im Beispielverhalten oder die gezielte Erzeugung von Fehlerbeispielen setzt bereits das Wissen um die Regeln für *korrektes* Verhalten voraus – der Lernansatz wäre damit ad absurdum geführt.

Eine geeignete Lösung des ersten Teilproblems hat folgende Vorteile gegenüber üblichen aktiven Testprozeduren, in denen ein Testgerät die Rolle eines Kommunikationspartners übernimmt:

- Ein passives Prüfverfahren kann unter genau den Bedingungen eingesetzt werden, unter denen auch der Betrieb des Kommunikationssystems stattfindet. Daher ist es relativ wahrscheinlich, daß gelegentliche Betriebsstörungen bei der Fehlersuche nicht verborgen bleiben. Wenn ein Problem beispielsweise nur auftritt, wenn ein seltener Zählerüberlauf stattfindet, wird ihn eine herkömmliche Testprozedur wahrscheinlich nicht aufdecken.

- Es kommt vor, daß die Kommunikation zwischen Komponenten verschiedener Hersteller gestört ist, obwohl beide Hersteller eine Zertifizierung ihrer Geräte vorweisen und zwischen Geräten desselben Herstellers keine Probleme auftreten. Dann ist eine Messung in genau der fehlerträchtigen Konfiguration erforderlich, um die verursachende Komponente zu ermitteln.

- In modernen Protokollen können gewisse, vom Standard offengelassene Parameter gelegentlich vom Benutzer oder Service-Personal nachträglich festgelegt werden. Probleme der Zusammenarbeit zweier Komponenten können dann auch auf unverträgliche Konfigurationseinstellungen zurückzuführen sein. Auch diese Fehlerquelle bleibt beim aktiven Testen verborgen.

- Wenn ein installiertes Kommunikationssystem zwar gelegentliche Störungen aufweist, seinen Betriebszweck aber noch im großen und ganzen erfüllen kann, ist es vorteilhaft, die Fehlersuche ohne Einfluß auf den Betrieb durchführen zu können. Ein aktiver Test einzelner Komponenten erfordert dagegen normalerweise eine Betriebsunterbrechung.

Die Lösung des zweiten Teilproblems erhält ihre praxisbezogene Rechtfertigung durch folgende Einsatzszenarien:

- Es wird später deutlich werden, daß die Entwicklung eines Prüfalgorithmus für ein konkretes Kommunikationsprotokoll selbst bei Vorliegen einer maschinenverständlichen Protokollspezifikation im allgemeinen nicht völlig automatisch erfolgen kann, sondern ein gewisses Maß an menschlicher Entwurfsarbeit erfordert. Wenn dazu nicht genügend Ressourcen zur Verfügung stehen, ist die Verwendung eines maschinellen Lernverfahrens ein denkbarer Ausweg. Immerhin gibt es heute eine sehr große Zahl von Protokollen und Protokollvarianten, so daß sich nicht für jede einzelne davon ein signifikanter Entwicklungsaufwand für eine Prüfmaschine lohnt.

- In der immer schnelleren Weiterentwicklung von Kommunikationssystemen und -standards beobachtet man häufig ein Nebeneinander von international standardisierten Protokollen und proprietären Entwicklungen, welche einen vorhandenen Bedarf möglichst schnell – das heißt vor der Konkurrenz und erst recht vor Abschluß eines meist langwierigen Standardisierungsprozesses – erfüllen sollen. Für derartige proprietäre Protokollvarianten sind formale Spezifikationen möglicherweise nicht oder nur zu einem hohen Preis an den Hersteller zu beschaffen. Ein Referenzgerät kann dann leichter und kostengünstiger verwendet werden.

Diese Arbeit wurde finanziert von der Siemens AG im Rahmen des Fachgremiums Meß- und Automatisierungstechnik innerhalb der Forschungskooperation zwischen Siemens und

der Technischen Universität Berlin. Projektpartner war die Abteilung AUT E33 D in Berlin, inzwischen abgelöst durch die Tektronix Berlin GmbH.

Diese Arbeit gliedert sich wie folgt: In Kapitel 2 wird der Stand der Technik in den Bereichen Protokoll-Engineering und Protokollmeßtechnik dargestellt sowie ein Überblick über bekannte maschinelle Lernverfahren gegeben. Auf dieser Grundlage erfolgt eine ausführliche, aber noch informelle Definition der oben beschriebenen beiden Teilaufgaben. In Kapitel 3 wird die erste Teilaufgabe formalisiert, analysiert und die zugehörige Lösung erarbeitet. Das Kapitel schließt mit der Wiedergabe von Erfahrungen aus der praktischen Anwendung des entwickelten Verfahrens. Kapitel 4 hat die zweite Teilaufgabe zum Inhalt, von der Formalisierung der Aufgabe über die Entwicklung zweier neuer Lernalgorithmen für unterschiedliche Teilprobleme und der Beschreibung des Gesamtverfahrens bis zu Ergebnissen der experimentellen Anwendung. In Kapitel 5 erfolgen eine vergleichende Bewertung beider Verfahren und der Anwendungsergebnisse mit einer kritischen Einstufung der Tauglichkeit für den praktischen Einsatz und eine Analyse, welche Ansatzpunkte für Weiterentwicklungen die automatisierte Verhaltensanalyse bietet, um Nachteile der entwickelten Verfahren zu überwinden. Kapitel 6 faßt die Ergebnisse der Arbeit zusammen.

Kapitel 2

Hintergrund und Aufgabe

In diesem Kapitel soll die wissenschaftlich-technische Grundlage dargestellt werden, auf der die neuen Ansätze dieser Arbeit aufbauen. Dies schließt sowohl den Stand der Technik im Bereich der Behandlung von Kommunikationsprotokollen ein als auch die bekannten Arten maschineller Lernverfahren und ihre prinzipielle Eignung für das Erlernen von Protokollregeln. Am Ende des Kapitels erfolgt, in Abgrenzung von der bekannten Technik, eine informelle Definition der im Rahmen der Arbeit bearbeiteten Aufgaben.

2.1 Stand der Technik

In dieser Arbeit geht es um die automatisierte, d. h. maschinelle Analyse von Kommunikationsprozessen. Wesentliche Grundlage hierzu sind die Modellierung und formale, also maschinell verarbeitbare Beschreibung von Kommunikationsprotokollen. Relevante Teilbereiche, die jeweils in Unterabschnitten behandelt werden, sind das international standardisierte Referenzmodell für digitale Kommunikation, der Einsatz formaler Methoden in den verschiedenen Phasen des Protokoll-Engineering und am Markt existierende Werkzeuge und Verfahren der Protokollmeßtechnik. Ein weiterer Unterabschnitt beschreibt die bekannten Ansätze zum maschinellen Lernen, die Berührungspunkte zur zweiten Teilaufgabe aufweisen, dem Erlernen von Protokollregeln.

2.1.1 OSI-Referenzmodell

Die zum Ansatz der offenen Kommunikation zwingend erforderliche internationale Standardisierung von Kommunikationsprozessen läuft ab unter dem Dach zahlreicher Organisationen, von denen folgende zwei die größte Bedeutung haben:

- Die ITU (*International Telecommunication Union*), früher CCITT (*Comité Consultatif International Télégraphiqe et Téléphonique*) ist der internationale Dachverband der nationalen Netzbetreiber-Gesellschaften. Ihre Telekommunikations-Sektion (ITU-T) beschließt *Empfehlungen*, die zwar nicht rechtlich verbindlich sind, aber de facto Standardcharakter haben.

- Die ISO (*International Organization for Standardization*), der Dachverband der nationalen Normungsinstitute, verabschiedet *Internationale Standards*. Auf dem Gebiet der Telekommunikation erfolgt eine enge Zusammenarbeit mit der ITU.

Gemeinsame Grundlage – und Ursprung der allgemeinen Terminologie – ist das 1984 verabschiedete Referenzmodell für offene Kommunikationssysteme (*open systems interconnection – basic reference model* [ISO84], typischerweise *OSI-Modell* genannt. Es teilt die Funktionalität einer digitalen Kommunikationsverbindung in sieben logische Schichten auf. Zentraler Gedanke dabei ist, daß jede Schicht einen nutzbaren *Dienst* erbringt und der nächsthöheren Schicht zur Verfügung stellt, indem sie den Dienst der nächsttieferen Schicht verwendet und unter Verwendung eigener Protokollmechanismen erweitert.

Die Standardisierung konkreter Kommunikationsprotokolle erfolgt im großen und ganzen schichtweise, so daß sich in der Implementierung ein *Protokollstapel* ergibt. Im Sinne des klassischen Abstraktionsgedankens im Software-Engineering sollte jede Protokollschicht von den konkreten Mechanismen abstrahieren, die tiefere Schichten zur Dienstrealisierung nutzen, d.h. von den konkret verwendeten tieferen Protokollen. Aufgrund besonderer Anforderungen in praktischen Kommunikationsumgebungen gelingt das jedoch nur teilweise, so daß zuweilen verfeinerte bzw. modifizierte Referenzmodelle zur Anwendung kommen, zum Beispiel in der modernen Breitbandkommunikation auf Basis der ATM-Vermittlungstechnik (*asynchronous transfer mode*, asynchroner Übertragungsmodus) [Sie94].

2.1.2 Formale Methoden

Im Zuge der Entwicklung offener Kommunikationssysteme entstand der Bedarf nach eindeutigen und vollständigen Beschreibungen für Kommunikationsprotokolle, d. h. nach *formalen Protokollspezifikationen*. Im Rahmen eines regelrechten Entwicklungsprogramms wurden daher zwischen 1981 und 1987 drei Spezifikationssprachen entwickelt und schließlich standardisiert [Hog89]: SDL (*specification and description language*) [CCI87], LOTOS (*language for the temporal ordering specification of observational behaviour*) [ISO87b] und Estelle [ISO87a]. Sie basieren auf zwei verschiedenen theoretischen Ansätzen, nämlich den *Prozeßalgebren* bzw. den *erweiterten endlichen Automaten*. Diese grundlegenden Kalküle und die auf ihnen aufbauenden Spezifikationssprachen werden in den folgenden Abschnitten kurz skizziert.

Prozeßalgebren

Dieses Kalkül zur Beschreibung nebenläufiger Prozesse geht zurück auf die Arbeiten [Mil80] und [Hoa85]. In den Prozeßalgebren wird ein *Verhalten* durch einen *Term* beschrieben, dessen operationale Semantik durch ein *markiertes Transitionssystem* (*labelled transition system*, LTS) wiedergegeben wird. Mathematisch ist ein LTS ein Tripel $(S, \xrightarrow{a}, A)$ aus:

- S, einer Menge von *Zuständen*,
- A, einer Menge von *Aktionen* (*actions*),

- $\xrightarrow{a} \subseteq S \times A \cup \{\tau\} \times S$, einer Menge von *Transitionen*.

Somit entspricht ein LTS strukturell einem endlichen Automaten. Jedes LTS kann eine Menge von Spuren (*traces*) zeigen, die aus einer über die Transitionsrelation $\xrightarrow{a}$ gebildeten Sequenz von Aktionen bestehen. τ ist eine ausgezeichnete Aktion, die *interne Aktion*, die von der Umgebung des Prozesses aus nicht beobachtbar ist.

Charakteristisch am Prozeßalgebra-Ansatz ist, daß das LTS durch einen Term definiert wird. In diesem können folgende Operatoren auftreten, die jedoch teilweise redundant sind, so daß in konkreten Prozeßalgebren eine Auswahl dieser Operatoren vorkommt:

- atomare Aktion aus $A \cup \{\tau\}$
- sequentielle Verkettung einer Aktion mit einem bereits definierten Verhalten
- Parallelisierung zweier Verhalten
- nichtdeterministische Auswahl zwischen zwei Verhalten
- Verbergen einer oder mehrerer Aktionen in einem Verhalten
- Umbenennen von Aktionen in einem Verhalten
- Verhaltensrekursion
- Termination
- Blockierung (*deadlock*)

Bei der Parallelisierung von Verhalten ist entscheidend, daß die Einzelverhalten über gleichnamige Aktionen synchronisiert werden, wovon τ ausgenommen ist.

Die so definierten LTSs werden üblicherweise in Form von Bäumen dargestellt, deren Kanten mit den Aktionen bezeichnet sind und deren Knoten sich mit Zuständen identifizieren lassen. Beispielsweise liefert in CCS (*calculus of communicating systems*, Kalkül für kommunizierende Systeme) [Mil80] der Term

$$coin.(tee\_button.tee + coffee\_button.coffee)$$

das in Figur A.1 dargestellte Modell eines Getränkeautomaten, ein beliebtes Beispiel in der Literatur über Prozeßalgebren.

Die Semantik der Verhaltensterme kann formal auf zwei Arten definiert werden: einerseits durch Ableitungsregeln, und andererseits, wie der Begriff Prozeß*algebra* nahelegt, durch ein Axiomensystem, mit dem sich Äquivalenzaussagen über Terme ableiten lassen. Da es mehr als 100 Äquivalenzbegriffe für Prozeßbeschreibungen gibt, ist diese Axiomatisierung alles andere als eindeutig. Ein Beispiel für eine Ableitungsregel ist die folgende Definition des Auswahloperators aus CCS:

$$\frac{E_1 \xrightarrow{\mu} E}{\begin{array}{l} E_1 + E_2 \xrightarrow{\mu} E \\ E_2 + E_1 \xrightarrow{\mu} E \end{array}}$$

Dies bedeutet: *Wenn* Verhalten E_1 mit der Aktion μ in Verhalten E übergeht, *dann* können sowohl Verhalten $E_1 + E_2$ als auch Verhalten $E_2 + E_1$ ebenfalls die Aktion μ ausführen und sich anschließend gemäß E verhalten.

Erweiterte endliche Automaten

Endliche Automaten (*finite state machines*, FSMs, für den Protokollsektor beispielsweise definiert in [vBP94]), sind der grundlegendste Ansatz zur Spezifikation von Systemverhalten und Kommunikationsprotokollen. Als operationaler Ansatz weisen endliche Automaten eine große Nähe zu möglichen Implementierungen auf.

Eine FSM A ist ein Sextupel $A = (S, X, Y, \delta, S_0, E)$ aus ([vBP94]):

- S, einer endlichen Menge von *Zuständen*,
- X, dem endlichen *Eingabealphabet*,
- Y, dem endlichen *Ausgabealphabet*,
- $\delta \subseteq S \times X \times Y \times S$, der *Transitionsrelation*,
- $S_0 \in S$, dem *Startzustand*,
- $E \subseteq S$, der Menge der *Endzustände*.

Die Transitionsrelation δ gibt an, welche Eingaben in einem Zustand welche Ausgaben hervorrufen können und in welchen Folgezustand der Automat anschließend wechselt.

In realen Protokollen werden häufig Sequenznummern und Bezeichner verwendet, die nicht übersichtlich mit endlichen Automaten zu beschreiben sind. Daher wird das Modell des endlichen Automaten um einen Datenteil aus Zustandsvariablen erweitert, womit man zum Modell des *erweiterten endlichen Automaten* (*extended finite state machine*, EFSM) kommt. Während der zugrundeliegende endliche Automat den *Kontrollflußaspekt* des Protokolls beschreibt, ist der *Datenflußaspekt* mit Hilfe der erweiternden Zustandsvariablen spezifiziert.

Mathematisch dient die aktuelle Wertbelegung der Zustandsvariablen dazu, die Menge der jeweils zulässigen Transitionen einzuschränken. Die Eingaben und Ausgaben werden ebenfalls um Datenfelder erweitert. Die Transitionen können Veränderungen der Variablenbelegung verursachen, in Abhängigkeit von den Datenfeldinhalten der zugeordneten Ein- oder Ausgabe. In der Literatur gibt es keine völlig einheitliche und originale formale Definition der EFSM. Man beachte aber, daß der einzige formale Unterschied zwischen FSMs und EFSMs in der Endlichkeit der Zustandsmenge liegt: Jede EFSM mit endlichen Wertebereichen aller Zustandsvariablen kann auch als äquivalente FSM modelliert werden. Nur bei Zustandsvariablen mit unendlichem Wertebereich ergibt sich tatsächlich eine größere Modellierungsmächtigkeit [Hog89].

LOTOS

LOTOS ist die 1987 von der ISO standardisierte Spezifikationssprache auf Prozeßalgebra-Basis. Das sogenannte *Basic-LOTOS* verwendet die in Abschnitt 2.1.2 aufgeführten Operatoren bis auf die Umbenennung, dafür kommen noch Parallelisierungsoperatoren dazu, die gar nicht bzw. nur auf einer gewissen Aktionsmenge synchronisieren.

Um den Datenflußaspekt realer System- und Protokollspezifikationen abzudecken, enthält das vollständige LOTOS außerdem ein Datenmodell mit Wertübergabe und Mustervergleich (*pattern matching*). Abstrakte Datentypen können mit Hilfe der algebraischen Spezifikationssprache ACT ONE definiert werden.

SDL

SDL ist die standardisierte Spezifikationsprache der ITU. Sie basiert auf dem Konzept der erweiterten endlichen Automaten. Datentypen können – wie bei LOTOS – in ACT ONE spezifiziert werden. Um verteilte Systeme beschreiben zu können, kann sich eine Systemspezifikation aus mehreren, über Kommunikationskanäle miteinander verbundenen erweiterten endlichen Automaten zusammensetzen, die in SDL *Prozesse* genannt werden. Die über die Kanäle ausgetauschten Signale werden in Warteschlangen aufbewahrt, es handelt sich also um *asynchrone* Kommunikation.

Als Besonderheit verfügt SDL über zwei verschiedene Notationen, eine textuelle (SDL/PR, *printed representation*) und eine äquivalente grafische (SDL/GR, *graphical representation*). Damit kann für den jeweiligen Einsatzzweck – z.B. interaktive Eingabe gegenüber maschineller Verarbeitung – die jeweils geeignetere Darstellung Verwendung finden. Figur A.2 zeigt ein Beispiel für eine einzelne Transition in der grafischen Notation SDL/GR: Nach Empfang des Signals *ConReq* im Zustand *Idle* wird die Zustandsvariable *c* inkrementiert, das Signal *ConAck* gesendet, und der Prozeß wechselt in den Zustand *Connecting*.

In SDL gibt es auch Möglichkeiten zur Behandlung von Zeit. Es lassen sich Zeitgeber definieren, die besondere Signale in der Warteschlange des Prozesses ablegen, sobald sie ablaufen. Nach dem auch als SDL-87 bezeichneten Standard [CCI87] wurde 1992 eine Erweiterung von SDL um objektorientierte Konzepte standardisiert [FO94].

Estelle

Estelle ist die automatenbasierte Spezifikationssprache der ISO. Konzeptionell unterscheidet sich Estelle nicht wesentlich von SDL. Es können ebenfalls mehrere kommunizierende Automaten definiert werden, die hier *Module* heißen.

Estelle ist syntaktisch stark an Pascal angelehnt und übernimmt dessen Datenmodell. Datentypen werden also nicht, wie in SDL, über algebraische Spezifikationen, sondern über ausführbare Implementierungen definiert. Eine graphische Repräsentation wie für SDL ist für Estelle nicht standardisiert.

Nachrichtensyntax

Zur vollständigen Beschreibung eines Kommunikationsprotokolls gehört neben der Spezifikation des Verhaltens auch die exakte Beschreibung der einzelnen Nachrichten, die im Rahmen des Protokolls zwischen den Kommunikationspartnern ausgetauscht werden. Diese Nachrichten werden als *Protokolldateneinheiten* (*protocol data units*, PDUs) bezeichnet.

Aus Sicht des Protokollentwurfs enthalten die PDUs Daten bestimmter Datentypen. So kommen häufig ganze Zahlen z. B. für Sequenznummern vor, Aufzählungstypen für die Unterscheidung verschiedener Arten von Nachrichten oder Zeichenketten für Adressierungsinformationen. Zusammen ergibt sich so eine Art Signatur des Protokolls, die als *abstrakte Protokollsyntax* bezeichnet wird.

Bei der Übertragung der PDUs wird eine Kodierung in eine Bitfolge benötigt, um die PDUs an die nächstniedrigere Protokollschicht übergeben zu können. Diese Kodierung heißt *konkrete Transfersyntax* der Protokollnachrichten und muß die korrekte Wiederherstellung der abstrakten Nachrichtenstruktur beim Empfänger gewährleisten.

Bei zahlreichen Protokollstandards werden die abstrakte PDU-Syntax und die konkrete Transfersyntax nicht streng getrennt, sondern – beispielsweise durch schematische Darstellung der Nachrichten auf Bitebene – gleichzeitig definiert, Beispiele dafür sind [CCI88, IT94a, IT94b]. Die ITU hat aber auch eine Beschreibungssprache, die *abstract syntax notation one* (*abstrakte Syntaxnotation eins*, ASN.1) standardisiert. ASN.1 definiert sowohl Datentypen für die abstrakte Syntax als auch eine konkrete Transfersyntax zur Übertragung dieser Datentypen als Bitstrom.

2.1.3 Protokoll-Engineering

Die in Abschnitt 2.1.2 aufgeführten Methoden und Sprachen finden Einsatz in verschiedenen Phasen der Protokollentwicklung. Dieser Abschnitt stellt die üblichen Teilaufgaben des Protokoll-Engineering vor und beschreibt die Fähigkeiten existierender Ansätze und Werkzeuge, die in den einzelnen Bereichen verwendet werden.

Spezifikation

Der primäre Einsatzzweck formaler Sprachen im Protokollbereich ist die eindeutige Beschreibung, also Spezifikation eines Protokolls. Dabei muß unterschieden werden zwischen der Spezifikation des *Dienstes*, den das Protokoll aus Sicht des Benutzers erbringt, und der Spezifikation des *Protokolls* selbst, mit dessen Hilfe dieser Dienst erbracht werden soll [Hog89]. Der Dienst wird mit Hilfe der *Dienstprimitiven* beschrieben, die an den Endpunkten der Kommunikationsstrecke (*Dienstzugangspunkte*, engl. *service access points*, SAPs) zur Verfügung stehen. Das Protokoll dagegen beinhaltet die Umsetzung der Dienstprimitiven in PDUs an einem einzelnen Kommunikationsendpunkt, einer Protokoll-*Instanz*.

Erst der Einsatz formaler Spezifikationssprachen erlaubt eine widerspruchsfreie und eindeutige Beschreibung von Diensten und Protokollen. Aus diesem Grund ist auch die ITU dazu übergegangen, in ihren Empfehlungen der formalen Protokollspezifikation Vorrang vor

der früher ausschlaggebenden natürlichsprachlichen Beschreibung einzuräumen. Während z. B. das Breitband-ISDN-Signalisierungsprotokoll Q.2931 in [IT94b] noch mit einer nur als Ergänzung deklarierten SDL-Spezifikation beschrieben ist, gilt im Standard des zugehörigen Schicht-2-Protokolls SSCOP (*service specific connection-oriented procedure*, dienstspezifische verbindungsorientierte Prozedur) in [IT94a] die SDL-Spezifikation bereits als ausschlaggebend, wo keine Eindeutigkeit herrscht.

Validierung

Die Validierung bestimmter Eigenschaften eines Protokolls ist eines der ältesten Motive für den Einsatz formaler Spezifikationen. v. Bochmann nennt in [Boc78] folgende wünschenswerte Eigenschaften einer Protokollspezifikation im Automatenkalkül:

Erreichbarkeit Alle Zustände des Automaten können über Folgen von Zustandsübergängen tatsächlich erreicht werden.

Verklemmungsfreiheit Das Gesamtsystem kann in keinen Zustand gelangen, aus dem kein weiterer Zustandsübergang möglich ist.

Lebendigkeit Ein Zustand oder Zustandsübergang ist lebendig, wenn er von jedem erreichbaren Systemzustand aus erreicht werden kann.

Konformitätstest

Konformitätstest (*conformance test*) nennt man die Prüfung, ob eine Protokollimplementierung sich gemäß der Protokollspezifikation verhält. Das Verfahren hierzu ist der aktive *Protokolltest*. Hier übernimmt ein dediziertes Testgerät die Aufgabe des oder der Kommunikationspartner der zu testenden Implementierung (*implementation under test*, IUT).

Der Konformitätstest hat eine besondere praktische Bedeutung als Abnahmetest für Implementierungen, die innerhalb oder in Verbindung mit öffentlichen Kommunikationsnetzen betrieben werden sollen und daher einer Zertifizierungspflicht unterliegen. Aus diesem Grund hat die ISO eine Methodologie und Nomenklatur für den Konformitätstest definiert und als Internationalen Standard IS 9646 [ISO91] verabschiedet. Innerhalb dieses Standards wird auch die Testbeschreibungssprache TTCN (*tree and tabular combined notation*, kombinierte Baum- und Tabellennotation) definiert. In [TKB91] erfolgt eine formalisierte Darstellung von Konzepten und Begriffen aus dem IS 9646, im wesentlichen aufbauend auf dem Konzept von markierten Transitionssystemen (LTS).

Eine *Testsuite* für einen Konformitätstest verfolgt mehrere *Testzwecke* (*test purposes*) und wird in *Testgruppen* (*test groups*) eingeteilt, die aus einzelnen *Testfällen* (*test cases*) bestehen. Ein Testfall, wie er sich in TTCN beschreiben läßt, besteht aus ein oder mehreren Eingaben vom Tester an die IUT und baumförmigen Verzweigungen in Abhängigkeit von der oder den Antworten der IUT. An den Blättern dieses Baumes stehen *Testurteile* (*verdicts*), und zwar [BG92]:

- *passiert* (*pass*) für korrektes Verhalten,

- *fehlgeschlagen* (*fail*) für nicht korrektes Verhalten und
- *unklar* (*inconclusive*) für ein Verhalten, das sich weder eindeutig als korrekt noch als inkorrekt einstufen läßt.

Je nach den Zugangsmöglichkeiten zur IUT kommen mehrere Testmethoden in Frage. Figur A.3 zeigt ein Beispiel für eine Testkonfiguration, in der der Tester entfernt über das Kommunikationsmedium mit der IUT verbunden ist und über eine zusätzliche, nur dem Test dienende Kommunikationsverbindung einen *oberen Tester* (*upper tester*, UT) steuert, der für die über den Dienstzugangspunkt der IUT ausgetauschten Dienstprimitiven zuständig ist.

Das in der Protokolltheorie am ausgiebigsten in der Literatur behandelte Thema ist die automatische Erzeugung von Testfällen und -suiten aus formalen Protokollspezifikationen. Beispiele für Arbeiten zu diesem Thema sind [Bri88, LL91, A+88]. Bochmann und Petrenko geben in [vBP94] einen umfassenden Überblick über das Problem des Protokolltests und seine Wechselwirkungen mit der unterliegenden formalen Protokollspezifikation. Je nach Art der Spezifikation lassen sich verschiedene Kriterien zum Grad der Testabdeckung bzw. Fehlererfassung (*test coverage* bzw. *fault coverage*) durch eine – vorzugsweise automatisch erzeugte – Testsuite definieren. Das Ziel der *vollständigen Fehlererfassung* bezieht sich dabei auf ein *Fehlermodell* (*fault model*), in dem alle fehlerhaften Implementierungen zusammengefaßt sind, deren Erkennung durch eine Testsuite angestrebt wird. Übliche Annahmen zur Bildung von Fehlermodellen sind eine obere Schranke für die Zustandsanzahl im FSM-Modell der Implementierung oder eine Beschränkung auf Ausgabefehler bei ansonsten korrekt implementierter FSM [vBP94].

Im Automatenmodell stößt das Ziel einer vollständigen Fehlererfassung auf Schwierigkeiten, sobald die Spezifikation Nichtdeterminismus enthält: Wann immer die Implementierung eine nicht von außen beeinflußbare Entscheidung treffen kann, welche nächste Ausgabe von einem Zustand aus erfolgt, lassen sich in trivialer Weise Automaten mit Zuständen konstruieren, deren Erreichen offensichtlich von keiner endlichen Testsuite erzwungen werden kann. Folgezustände und -transitionen eines solchen nichtdeterminismusbehafteten Zustands lassen sich also durch Testen niemals sicher überprüfen. Dies ist die auf den Protokolltest bezogene Begründung der wohlbekannten Tatsache, daß durch Testen niemals die Abwesenheit von Fehlern bewiesen werden kann, nur manchmal ihre Anwesenheit. Im Kontext der Prozeßalgebra und der LTSs lassen sich bessere formale Resultate beweisen (z. B. [FvB91]), weil dieses Paradigma von synchroner Kommunikation ausgeht: Ein Testerprozeß hat die Möglichkeit, alternative Aktionen der IUT selektiv zu blockieren. Diese Annahme ist für reale Kommunikationssysteme aber völlig unrealistisch, weil die physikalische Kommunikation auf der Schicht 1 des OSI-Modells immer asynchron erfolgt.

Leistungsbewertung

Die bisher behandelten Aufgabestellungen und Methoden betreffen ausschließlich die Modellierung und Analyse des *logischen* Verhaltens von Kommunikationssystemen. Auf der anderen Seite besteht ein Bedarf an Modellierungswerkzeugen, die es erlauben, *Leistungsaspekte* von Kommunikationssystemen anhand eines Modells zu ermitteln. Solche Leistungsaspekte werden teilweise durch stochastische und teilweise durch deterministische Kenngrößen dargestellt. Zu den stochastischen Kenngrößen gehören beispielsweise:

- Der erreichbare mittlere *Durchsatz* von Nachrichten durch bestimmte Systemkomponenten.
- Die *Wahrscheinlichkeit*, daß eine bestimmte fehlerhafte oder unerwünschte Situation auftritt. Dafür kommen beispielsweise Pufferüberläufe, Überschreitungen von Zeitschranken oder Kollisionen beim Zugriff auf Rundsendemedien wie beim Ethernet in Frage.
- Mittlere *Verzögerungen* bestimmter Systemantworten.
- *Erwartungswerte* für Systemgrößen wie Pufferfüllstände.

Deterministische Kenngrößen erhält man entweder als Extremwerte der beschriebenen stochastischen Größen oder in vollständig deterministischen Leistungsmodellen. Beispiele sind:

- Der *maximale* Füllstand eines Puffers, wichtig für dessen Dimensionierung.
- Die *maximale* Antwortzeit auf eine Nachricht oder ein Ereignis, wichtig beim Entwurf von Echtzeitsystemen.
- Die tatsächliche Laufzeit einer Nachricht im Falle deterministischer Modellierung.

Im Gegensatz zu den beschriebenen logischen Modellierungsansätzen brauchen Leistungsmodelle nicht alle Feinheiten der Protokollabwicklung zu beinhalten. Statt dessen müssen relevante Zeitdauern für Zustandswechsel, im Extremfall als Wahrscheinlichkeitsverteilungen dieser Zeiten, und Auswahlwahrscheinlichkeiten für nichtdeterministische Entscheidungen modelliert werden. Die folgende Liste nennt eine Reihe von Modellierungswerkzeugen zur Leistunganalyse:

- *Warteschlangenmodelle* sind Modellierungen stochastischer Prozesse, bei denen Warteschlangen mit einer bestimmten Bedienstrategie und ein Bedienprozeß mit einer bestimmten Verteilungsfunktion der Bedienzeit als Elemente dienen.
- *Stochastische Petri-Netze* [Mol82] sind Erweiterungen von Petrinetzen [Rei85] um Zeitkonzepte. Damit lassen sich verteilte und nebenläufige Systeme in ihren Zeitaspekten grafisch modellieren. Viele Klassen stochastischer Petrinetze erlauben eine Leistungsbewertung nicht nur durch Simulation, sondern auch durch numerische Analyse des durch das Netz beschriebenen stochastischen Prozesses.
- *Zeiterweiterte Prozeßalgebren* sind Ergänzungen des in Abschnitt 2.1.2 beschriebenen Prozeßalgebra-Konzepts um Zeitaspekte.
- Spezielle *Simulationssprachen* erlauben die Beschreibung von verteilten und nebenläufigen Systemen auf einer Ebene, die näher bei den Programmiersprachen liegt. Das Verhalten einzelner Systemkomponenten läßt sich meist direkt als Programmfragment spezifizieren. Solche Modelle erlauben, weil sie sehr allgemeine Modellierung ermöglichen, nur noch die Simulation zur Ermittlung von Leistungskenngrößen.

Es ist heute ein offenes Problem, ob es möglich und sinnvoll ist, zur vollständigen logischen Spezifikation und zur Leistungsbewertung ein und dasselbe Modell zu verwenden. Ein solches Vorgehen hätte die Vorteile, daß die Modellierung nur einmal erfolgen muß und keine Inkonsistenzen zwischen beiden Modellierungssichten infolge von Modellierungsfehlern entstehen können. Sowohl die Simulation als auch die Analyse von Leistungsmodellen reagieren jedoch sehr empfindlich auf die Größe des Zustandsraums, den das Modell aufspannt. Daher erfolgt die Leistungsbewertung heute meist anhand von Modellen, die manuell auf die relevanten Aspekte reduziert sind und vom vollständigen logischen Ablauf abstrahieren. Eine solche Reduktion in formal korrekter Weise automatisch vornehmen zu können ist die Grundvoraussetzung, um die logische und die Leistungsanalyse schließlich zusammenzuführen.

Protokollmonitore

Für keines der bisher aufgeführten Konzepte trifft der in der Einleitung umrissene Einsatzkontext einer *passiven Messung im laufenden Kommunikationssystem* zu. Der Stand der Technik in diesem Bereich soll jetzt untersucht werden.

Seit geraumer Zeit sind Geräte kommerziell verfügbar, die uneinheitlich als *Protokolltester*, *Protokollmonitore* oder *Protokollanalysatoren* bezeichnet werden. Im folgenden soll der Begriff *Protokollmonitor* für diese Geräteklasse verwendet werden. Es handelt sich um transportable Rechner, die meist mittels spezieller Schnittstellenmodule und Software in der Lage sind, Messungen an ein oder mehreren Beobachtungspunkten in einem Kommunikationssystem vorzunehmen [McC90, HMS91, SMM98]. Folgende Funktionsmerkmale sind typisch für Protokollmonitore:

- Automatische *Dekodierung* und menschenlesbare Anzeige der Nachrichteninhalte. Statt eines binär oder hexadezimal dargestellten Ur-Datenstroms sieht der Benutzer die Inhalte der einzelnen Datenfelder der Nachrichten mit den Feldbezeichnungen und in einer passenden numerischen oder textuellen Repräsentation. Diese Aufgabe entspricht der Rückübersetzung von der konkreten Transfersyntax des Protokolls in die abstrakte Syntax (vgl. Abschnitt 2.1.2).

- Erkennung und Anzeige von *Kodierungsfehlern*, d. h. von Fällen, in denen die Kodierungsregeln des angezeigten Protokolls für einzelne Nachrichten verletzt werden. Beispiele sind Bereichsüberschreitungen für numerische Feldinhalte oder das Fehlen von Datenfeldern, die zwingend vorgeschrieben sind.

- Setzen von *Filtern*, um nur bestimmte Nachrichten – z. B. nur die Nachrichten einer bestimmten Schicht oder nur einen Nachrichtentyp mit ganz bestimmten Feldinhalten – anzuzeigen. Solche Filter erlauben es, bestimmte Kommunikationsituationen schrittweise einzugrenzen und zu isolieren.

- *Statistiken* erzeugen, die das Aufkommen an Kommunikationsverkehr nach unterschiedlichen Kriterien anzeigen. Als Kriterien kommen die verschiedenen auf derselben Strecke laufenden Protokolle in Frage, verschiedene virtuelle Verbindungen, einzelne Nachrichtentypen oder Kombinationen davon.

Die Analyse des Kommunikationsverhaltens über eine Folge von Nachrichten hinweg gehört *nicht* zu den Fähigkeiten heutiger kommerzieller Protokollmonitore. Diese Aufgabe muß vom Benutzer eines Protokollmonitors *in Handarbeit* erledigt werden. Der Benutzer kann dabei lediglich auf die Funktionsmerkmale des Monitors zurückgreifen, beispielsweise Filter setzen und diese schrittweise einengen.

Spuranalyse

Die automatische, nachrichtenübergreifende Analyse eines Kommunikationsverhaltens wird in der Literatur als Spuranalyse (*trace analysis*) bezeichnet. Hierbei wird eine Teilmenge der *Interaktionspunkte* (*interaction points*, IPs) einer IUT *passiv* beobachtet, um eine *Spur* ihres Verhaltens aufzuzeichnen. Der Analysator greift in keiner Weise in den Kommunikationsvorgang ein. Er rekonstruiert die Folge interner Aktionen der IUT, die im Einklang stehen mit der Protokollspezifikation, mit den beobachteten Eingaben in die IUT und ihren beobachteten Ausgaben.

Im Vergleich zu den zuvor beschriebenen Themenkomplexen Spezifikation, Validierung, Verifikation, *aktiver* Protokolltest und Testgenerierung ist in der Literatur vergleichsweise wenig zum Thema Spuranalyse veröffentlicht worden.

Zwei der ersten Papiere, die einen Überblick über die Spuranalyse von Kommunikationsprotokollen liefern, sind [JvB83] und [UP86]. Diese Arbeiten konzentrieren sich zwar hauptsächlich auf die Validierung der Verfeinerungen von Spezifikationen, aber sie führen bereits grundlegende Begriffe und Lösungsvorschläge für das Testen von Protokollimplementierungen ein.

Eine Implementierung eines „passiven Monitors" – nach der Terminologie der vorliegenden Arbeit eines Spuranalysators – für die gleichzeitige Analyse mehrerer OSI-Protokollschichten wird in [CL91] beschrieben. Den Schwerpunkt bildet dort eine beispielhafte Anwendung des Analysators auf das X.25-Protokoll [CCI88]. Das beschriebene System basiert auf in C kodierten Zustandsmaschinen und erfordert ein vollständiges Zurücksetzen der Kommunikationsverbindung, um mit einem definierten Ausgangszustand seine Analyse beginnen zu können.

Die Arbeit [KCV92] präsentiert einen theoretischen Ansatz, der vom FSM-Kern einer in einer Untermenge von Estelle formulierten Spezifikation ausgeht und zunächst *alle* Folgen von internen Aktionen (*action paths*) aufzählt, die die Spezifikation zuläßt. Anschließend werden durch symbolische Auswertung alle Pfade herausgelöscht, die nur wegen der Zustandsvariablen oder Nachrichtenparameter ungültig sind. Dieses Verfahren ist vom theoretischen Standpunkt aus attraktiv, da es sowohl zur Spuranalyse als auch zur Testfallgenerierung verwendet werden könnte. Allerdings ergeben sich Aufwandsprobleme aus der Tatsache, daß hier mit ganzen Pfaden anstelle von Zuständen gearbeitet wird.

In [BvBDS91] wird Spuranalyse gegenüber LOTOS-Spezifikationen behandelt und das Analysewerkzeug *TETRA* vorgestellt. Seine Prolog-Implementierung des Spuranalysealgorithmus verwendet uninstantiierte Prolog-Variablen, um das Problem nichtdeterministischer Auswahlen und nichtdeterministischer Werterzeugung zu lösen. Dieser Ansatz beinhaltet keine Zeitbedingungen, weil das Original-LOTOS – ohne Zeiterweiterung wie beispielsweise in [Sch94] – keine Zeit modelliert. Außerdem ist die Auswertung von LOTOS- und Prolog-Konstrukten ineffizient.

Die Arbeit [EvB95] stellt den *Tango*-Übersetzer vor, der eine Estelle-Spezifikation mit einem einzelnen Modul und ohne Zeitbedingungen in einen protokollspezifischen Spuranalysator übersetzt. Dieser Ansatz ist sehr elegant, weil Estelle direkt verwendet wird. Die Arbeit spricht alle wesentlichen Aspekte einer realen Spuranalyse in Echtzeit, d. h. während der Kommunikation, an.

2.1.4 Lernverfahren

Im Laufe der Zeit sind immer mehr Verfahren entwickelt worden, die aus einem Satz von Beispielen, einer Trainingsmenge, maschinell eine Klassifikationsfunktion konstruieren, die charakteristische Eigenschaften der Trainingsmenge beschreibt und weitere, nicht in der Trainingsmenge enthaltene Beispiele geeignet klassifizieren soll. Dazu gehören so unterschiedliche Ansätze wie statistische Klassifikatoren, Abstandsklassifikatoren, Verfahren auf Basis neuronaler Netze, z. B. Backpropagation [RM86], Entscheidungsbaumverfahren [UW81] sowie das Konzept der *Induktiven Logischen Programmierung* (*inductive logic programming*, ILP) [BM95, LD94, L+96].

Nach Kenntnis des Autors ist in der Literatur bisher nicht von der Anwendung eines Lernverfahrens berichtet worden, um Regeln eines Kommunikationsprotokolls aus einer beobachteten Protokollabwicklung abzuleiten. Im folgenden Abschnitt werden daher einige grundlegende Eigenschaften dieser Lernaufgabe angesprochen, um einen Bezug zwischen den bekannten Verfahren und der hier betrachteten Anwendung herstellen zu können. Die Abschnitte danach geben ein bekanntes theoretisches Resultat zur Lösbarkeit dieser Aufgabe wieder und gehen auf die Eignung bekannter Verfahren ein.

Charakterisierung der Lernaufgabe

In Abschnitt 2.1.2 wurden Formalismen erläutert, die man zur Spezifikation von Protokollen verwendet. Aus der Kenntnis dieser Methoden läßt sich ableiten, daß das Erlernen eines Protokolls formal betrachtet das Erlernen einer *Sprache* darstellt, und zwar der Sprache der mit dem Protokoll vereinbaren Folgen beobachteter PDUs. Diese wird als *Spursprache* (*trace language*) des Protokolls bezeichnet und durch die Semantik des Prozeßalgebra- oder Automatenmodells der Protokollspezifikation originär definiert.

In der Einleitung wurde bereits angedeutet, daß die Protokollabläufe, aus denen die Protokollregeln erlernt werden sollen, ausschließlich *positive Beispiele* sein werden. Diese Forderung ist eine zwingende Folge des Anwendungskontextes: Um im Besitz negativer Beispiele zu sein, d. h. von Beispielen mit Protokollverstößen, oder um einen beliebigen Beispielverkehr in korrekte und fehlerhafte Abläufe aufteilen zu können, muß man die Spezifikation des korrekten Protokollverhaltens bereits zur Verfügung haben. Dann macht es aber keinen Sinn mehr, ein Lernverfahren einzusetzen. Ohne Kenntnis der Spezifikation ist es aber die naheliegendste Annahme, daß man eine Referenzstrecke mit vollständig protokollkonformer Kommunikation zur Verfügung hat. Hieraus folgt, daß das Lernverfahren allein mit positiven Beispielen auskommen muß. In der Literatur wird unterschieden zwischen *überwachtem* und *unüberwachtem* Lernen (*supervised* bzw. *unsupervised learning*) [HKP91]. Im Fall des überwachten Lernens erhält der Lerner – von einer gedachten Lehrerinstanz – die Klasseneinteilung der

Lehrbeispiele mitgeteilt, z.B. eine Einteilung in Positiv- und Negativfälle. Im zweiten Fall, der damit der vorliegenden Anwendung entspricht, sieht der Lerner ausschließlich die Beispiele ohne jeden Hinweis einer Klassenzugehörigkeit.

In einigen Lernansätzen wird das Fehlen konkreter Negativbeispiele durch die Annahme einer *abgeschlossenen Welt* (*closed world assumption*, CWA) kompensiert. Die CWA sagt aus, daß alle nicht in der Trainingsmenge enthaltenen Beispiele als Negativbeispiele gelten sollen, also

$$\bar{E} = U \setminus E,$$

wenn $\bar{E}$ die Menge der negativen Beispiele, E die der positiven Beispiele und U das Universum der Klassifikationsaufgabe symbolisieren. Die CWA ist beim Erlernen eines Protokolls jedoch unbrauchbar, weil aufgrund der Komplexität realer Protokolle im allgemeinen nicht *alle möglichen* korrekten Protokollabläufe in einer Trainingsmenge vorkommen können. Dies sieht man leicht ein, wenn man beachtet, daß beispielsweise im Protokoll Q.2110 [IT94a] pro Protokollinstanz u. a. 4 Zählervariablen mit jeweils einem Wertebereich der Mächtigkeit 2^{24} vorkommen. Daraus folgt, daß sich die Lernaufgabe als *generalisierendes Lernen aus positiven Beispielen* charakterisieren läßt.

In Protokollen kommen oft Zählervariablen und Sequenznummern vor, über die Bestätigungs- und Wiederholungsmechanismen definiert werden (beispielsweise [IT94a, CCI88]). Exemplarisch soll hier der Fall zweier beobachteter PDUs mit jeweils einem numerischen Datenfeld, geschrieben $A(x)$ und $B(y)$, betrachtet werden. In solchen Fällen sagt der numerische Inhalt eines PDU-Datenfeldes allein nichts über die Kontextkorrektheit der PDU aus, eine Regel der Form

$$x = c \quad (c \in \mathbb{N})$$

ist also wertlos. Selbst eine Konjunktion der Form

$$x = c_1 \wedge y = c_2 \quad (c_1, c_2 \in \mathbb{N})$$

hilft nicht weiter, es sei denn, es existiert eine eigene Regel für *jeden* Wert aus dem Wertebereich von x bzw. y, im obigen Beispiel der Q.2110 also vielleicht 2^{24} Regeln. Denn der wirkliche, vom Protokoll verlangte Zusammenhang könnte lauten:

$$y = x + 1 \quad \Leftrightarrow \quad y - x = 1$$

Es ist demnach erforderlich, daß das Lernverfahren selbst *arithmetische Terme* – hier etwa $(y - x)$ – bildet, um in den Regeln darauf Bezug zu nehmen.

Lösbarkeit

Das in Abschnitt 2.1.4 eingeführte Sprachlernparadigma wird in der Literatur als *induktive Inferenz* (*inductive inference*) [Sol64a, Sol64b] bezeichnet. Ein sehr frühes theoretisches Resultat, das die algorithmische Lösbarkeit dieses Problems betrifft, stammt von E. M. Gold aus dem Jahr 1967 [Gol67]: Selbst reguläre Sprachen können nicht *im Limit* aus rein positivem Beispieltext gelernt werden. Weil dieses Ergebnis so wichtig ist, soll es kurz formal vorgestellt werden.

Definition 1 *Gegeben ein endliches Alphabet Σ, eine Familie Λ von Sprachen $L \in \Sigma^*$ und ein Algorithmus α, der eine Folge von Wörtern $x_i \in \Sigma^*$ als Eingabe erhält und eine Grammatik G als Hypothese für die Sprache ausgibt, zu der die x_i gehören:*

$$\alpha(x_1, \ldots, x_t) = G \qquad (x_1, \ldots, x_t \in L_G)$$

Man sagt, α identifiziert eine Sprache L im Limit, *wenn alle $x_1, \ldots, x_t \in L$ und für eine Schrittnummer t_0 gilt:*

$$\forall t \geq t_0 \quad \alpha(x_1, \ldots, x_t) = G = \text{const.} \wedge L_G = L$$

Die Familie Λ von Sprachen heißt identifizierbar im Limit, *wenn es einen Algorithmus α gibt, der jedes $L \in \Lambda$ im Limit identifiziert.*

Die x_i werden auch als *unendlicher positiver Text* für L bezeichnet. Der Algorithmus α sieht einen immer länger werdenden Text der zu lernenden Sprache und darf seine Hypothese nur endlich oft ändern, bis sie gegen eine Grammatik für L konvergiert.

Gold hat gezeigt, daß noch nicht einmal die Klasse aller endlichen Sprachen plus einer einzigen unendlichen im Limit identifizierbar ist. Die regulären Sprachen können also erst recht nicht aus einem positiven Text gelernt werden. Anschaulich ist dieses Ergebnis darin begründet, daß die Anzahl der Zustände des endlichen Automaten, der die zu lernende Sprache modelliert, nicht begrenzt ist. Daher existiert zu *jeder* noch nicht beobachteten Symbolsequenz immer auch eine Automatenhypothese, die auch diese Sequenz akzeptieren *könnte*.

Eignung bekannter Verfahren

Aus verschiedenen der in Abschnitt 2.1.4 genannten Gründe eignet sich kein bekanntes Lernverfahren unmittelbar für das Erlernen eines Kommunikationsprotokolls.

Das *Backpropagation*-Verfahren (BP) [RM86] als bekanntester Vertreter der konnektionistischen Ansätze ist im Kern ein nichtlineares Regressionsverfahren, das eine recht effiziente Gradientensuche im Parameterraum erlaubt. Für Klassifikationsaufgaben kann BP jedoch nur im überwachten Lernen verwendet werden, da im Falle einer konstant „1" oder „korrekt" lautenden Zielfunktion, quasi ungeachtet der Trainingsmenge, auch nur die Ausgabe „korrekt" erlernt wird. Dieser Sachverhalt ist auf weitere konnektionistische Verfahren vom Typ des überwachten Lernens übertragbar, z. B. auf Regressionsverfahren mit radialen Basisfunktionen [HKP91].

Ein konnektionistisches Verfahren unüberwachten Lernens sind die *Merkmalskarten* (*feature maps*) [HKP91], bei denen eine topologische Abbildung eines mit einem Ähnlichkeitsmaß versehenen Merkmalsraums in einen strukturierten Repräsentantenraum gelernt wird. Bei diesem Verfahren können – wie bei allen anderen Methoden, die auf einem vorgegebenen Ähnlichkeitsmaß zur Cluster-Bildung beruhen – arithmetische Zusammenhänge nicht im erforderlichen Umfang erfaßt werden.

Verfahren aus der Klasse der *Entscheidungsbäume* (*decision trees*) bilden Klassifikationsregeln als einen Baum logischer Tests, eine Darstellungsform des Lernresultats, die kompakt,

effizient auswertbar und überdies auch für Menschen nachvollziehbar ist [UW81]. Diese Verfahren benötigen durchweg Klasseninformation in der Trainingsmenge, und neu berechnete Attribute werden auch nicht verwendet.

Näher an die vorliegende Problemstellung kommen die Verfahren aus dem Bereich der *Induktiven logischen Programmierung* (*inductive logic programming*, ILP). Statt einer Partitionierung des Merkmalsraums nach einem relativ unflexiblen Muster wird hier ein logisches Programm, meist in Hornklausellogik, gelernt. Der formale Grundansatz (nach [LD94]) ist der folgende:

> Gegeben das Hintergrundwissen B, eine Menge von Positivbeispielen E und eine Menge von Negativbeispielen $\bar{E}$, finde eine Hypothese H (ein logisches Programm, z. B. in Hornklausellogik), so daß
>
> 1. $B \wedge H \models E$
> 2. $\forall x \in \bar{E} : B \wedge H \not\models x$

Schon aus dieser Formalisierung ergibt sich die Rolle der Negativbeispiele. ILP-Verfahren sind überwiegend auf Negativbeispiele oder die CWA angewiesen. Außerdem eignen sie sich fast ausschließlich für symbolische Welten, wo Prädikate aus dem Hintergrundwissen auf eine überschaubare Zahl von Atomen angewandt werden können. Kein bekanntes ILP-Verfahren kann mit beliebigen Prädikaten über numerische Attribute operieren, erst recht nicht arithmetische Operationen konstruieren.

2.2 Informelle Aufgabendefinition

Nachdem der Stand der Technik beleuchtet wurde, gibt dieser Abschnitt eine Beschreibung der Aufgaben, die in dieser Arbeit gelöst werden sollen. Dies erfolgt getrennt nach den Teilaufgaben Spuranalyse und Protokoll-Lernen.

2.2.1 Spuranalyse

Hier wird die Aufgabenstellung des Aufgabenteils Spuranalyse präzisiert, also die passive Überprüfung eines Kommunikationsverhaltens anhand einer *vorgegebenen* Protokollbeschreibung. Dazu gehören das Einsatzszenario für ein entsprechendes Prüfgerät, seine Funktionalität im Hinblick auf die zu erhaltenen Ergebnisse der Analyse und eine Betrachtung der besonderen Anforderungen, die beim Lösungsansatz beachtet werden müssen.

Einsatzszenario

Das physische Einsatzszenario des Spuranalysators im Sinne dieser Arbeit ist in Figur A.4 dargestellt: Der Analysator ist mit dem Kommunikationsmedium verbunden und kann die darüber zwischen den Kommunikationspartnern A und B ausgetauschten PDUs mit ihren Datenfeldern mitlesen. Er hat jedoch keine Möglichkeit, in irgendeiner Weise in den Ablauf der Kommunikation einzugreifen.

Nur System A ist in der Figur als IUT ausgewiesen, weil der Analysator konzeptionell nur die Richtigkeit des Verhaltens *einer* Partnerinstanz überwacht. Das heißt, daß im Falle von fehlerhaften Protokollprozeduren in jedem Fall geprüft wird, ob Partner A auf potentiell fehlerhaftes Verhalten von B richtig, also protokollkonform, reagiert. Wenn dagegen die *globale* Protokollkonformität des beobachteten Verkehrs geprüft würde, also nur die Gutfallabläufe des Protokolls als fehlerfrei eingestuft würden, könnte bei einer Abweichung der verursachende Kommunikationspartner nicht automatisch festgestellt werden. Die Festlegung des Verfahrens auf eine *beobachtete Instanz* oder IUT stellt keinerlei Einschränkung dar, weil der Analysator prinzipiell zwei oder mehr Prüfprozesse nebenläufig ausführen kann, um alle beteiligten Kommunikationspartner auf korrektes Verhalten zu überwachen. Folglich ist ein Ansatz mit *einer* IUT der allgemeinere und daher der einzig sinnvolle.

Figur A.5 zeigt die logische Einbindung des Analysealgorithmus in den Kommunikationsprozeß: Der Analysealgorithmus erhält die PDUs einer Protokollschicht n, $n \geq 1$, aus dem Protokollstapel der IUT als Eingabe. Er prüft die Abwicklung dieses Protokolls und der Protokolle der unmittelbar darüberliegenden Protokollschichten, insgesamt k Protokolle, $k \geq 1$. Der Analysator erhält dagegen keinerlei Information über die an den Dienstzugangspunkten (SAPs) ausgetauschten Primitiven.

In dieser Arbeit wird nicht betrachtet, wie man die Anbindung des Analysators an das Kommunikationsmedium realisieren könnte. Diese Aufgabe stellt kein wesentliches Problem dar, sie wird z. B. von den verfügbaren Protokollmonitoren gelöst, die in Abschnitt 2.1.3 beschrieben wurden.

Als Grundlage der Überprüfung muß der Analysator über ein Modell für das korrekte Kommunikationsverhalten der IUT verfügen. Dieses Modell soll hier vom Informationsgehalt her einer formalen Protokollspezifikation gemäß dem Protokollstandard entsprechen. Der genaue Ansatz der Protokollmodellierung wird in Kapitel 3 erläutert und begründet.

Es ist klar, daß die Überprüfung einer beliebigen Schicht nur dann sinnvolle Ergebnisse liefert, wenn *alle* darunterliegenden Schichten ihren vorgesehenen Dienst erbringen. Wenn beispielsweise PDUs einer überwachten Schicht ausbleiben, weil sie bei der Übertragung durch die unterliegenden Schichten zwischen der IUT und dem Analysator verlorengehen, kann dies nicht von einer Fehlfunktion in der überwachten Protokollschicht unterschieden werden. Dieser Sachverhalt ist besonders bedeutend im Hinblick auf Bitfehler in Schicht 1, der physikalischen Schicht, weil diese – wenn auch mit geringer Wahrscheinlichkeit – in jedem einwandfrei arbeitenden Kommunikationssystem auftreten können und dürfen. Solche Bitfehler erscheinen als Fehlfunktion der Schicht 1 und aller Schichten unterhalb der ersten Schicht, die die gesicherte Datenübertragung z. B. durch Wiederholung der gestörten PDUs gewährleistet. Von wirklichen Fehlfunktionen müssen solche Fälle durch ihre geringe Häufigkeit unterschieden werden.

Funktionalität

Die wesentliche Aufgabe des Spuranalysators ist, *Fehlerereignisse* im Kommunikationsablauf der überwachten Schichten zu erkennen und zu melden. Hier erfolgt zunächst eine – noch weitgehend informelle – Definition, was darunter zu verstehen sein soll:

Definition 2 *Ein* Fehlerereignis *im beobachteten Kommunikationsverhalten einer Protokollinstanz ist der früheste Zeitpunkt t_e einer zur Zeit t_0 begonnenen Beobachtung, für den die Beobachtung im Zeitraum $[t_0; t_e]$* nicht *in der Menge der vom zugrundeliegenden Protokoll zugelassenen Ereignisfolgen enthalten ist.*

Aus der Definition folgt, daß es pro Beginn einer Beobachtung nur ein Fehlerereignis geben kann. Nachdem ein Fehlerereignis erkannt worden ist, muß also ein neuer – späterer – Startzeitpunkt gewählt werden, der allerdings nicht unbedingt nach dem Fehlerzeitpunkt liegen muß. Diese Konsequenz eines neuen Analysebeginns ist zweckmäßig, weil es nach dem Erkennen eines Protokollverstoßes keinerlei Anhaltspunkt gibt, welches weitere Protokollverhalten korrekt oder nicht korrekt wäre. Figur A.6 gibt den Zusammenhang zwischen Beobachtungen und Fehlerereignissen nach Definition 2 beispielhaft wieder.

Ein konkreter Fehler innerhalb eines Kommunikationssystems, egal welcher der in der Einleitung genannten Arten, führt im Regelfall zu immer wiederkehrenden Fehlerereignissen, je länger die Kommunikation beobachtet wird. Eine große Zahl von gemeldeten Fehlerereignissen ist für den Benutzer des Analysators aber nicht hilfreich, weil er nach dem konkreten verursachenden Fehler sucht, der beseitigt werden soll, und nicht nach den einzelnen Fehlerereignissen. Es besteht also die zusätzliche Aufgabe, erkannte Fehlerereignisse so zu *klassifizieren*, daß jede Klasse von Fehlerereignissen auf denselben konkreten Fehler im System zurückzuführen ist. Die Klassifikation kann nur danach erfolgen, welche Fehlerereignisse sich so *ähnlich* sind, daß sie – vermutlich – vom selben Fehler verursacht wurden. Ein unmittelbarer Rückschluß auf die verursachenden Fehler könnte nur erfolgen, wenn der Analysator über ein Modell der *Systemimplementierung* verfügt, das eine Verfeinerung der Protokollspezifikation darstellt. Ein solcher Ansatz wird in dieser Arbeit nicht verfolgt, und zwar aus folgenden Gründen:

- Ein formales Implementierungsmodell wäre erheblich komplizierter als die Protokollspezifikation und sehr viel schwieriger zu erstellen bzw. zu beschaffen.
- Ein formales Implementierungsmodell wäre nur für einzelne Zielsysteme verwendbar, so daß dem erhöhten Aufwand ein verringerter Nutzen hinsichtlich der Zahl der Anwendungsfälle gegenübersteht.

Eine bereits erwähnte Eigenschaft vieler Protokollstandards ist die, daß der Standard Parameter und Optionen vorsieht, die erst bei einer konkreten Implementierung oder Anwendung des Protokolls festgelegt werden müssen. Die jeweils gewählten Festlegungen sind beispielsweise im *protocol implementation and conformance statement* (PICS, [ISO91]) aufgeführt. Eine sinnvolle Zusatzfunktion für den Spuranalysator besteht daher darin, diese Parameter, sofern sie dem Anwender nicht bekannt sind, aus der Beobachtung zu ermitteln bzw. zu schätzen. Man muß dabei beachten, daß eine solche Ermittlung keinen vollwertigen Ersatz darstellen kann für die *Kenntnis* der PICS-Parameter, weil erstere im *Ist*-Verhalten stattfindet und letztere das *Soll*-Verhalten charakterisieren.

Obwohl einzelne Fehlerereignisse nach den bisherigen Erkenntnissen keinen Rückschluß auf verursachende Fehler erlauben, lassen sich dennoch weitere Informationen aus der Häufigkeit von Fehlern und Fehlersituationen im Verlauf der Beobachtung gewinnen. So ist eine Klassifikation nach *transienten*, *permanenten* und *intermittierenden* Fehler möglich:

Transiente Fehler treten nur einmalig auf. Später beobachtet man in der gleichen Kommunikationssituation das korrekte Verhalten.

Permanente Fehler führen wiederholt zu Fehlerereignissen, und zwar jedesmal, wenn die Kommunikationssituation auftritt, in der sich der verursachende Fehler des Systems auswirkt.

Intermittierende Fehler treten zwar wiederholt, aber nicht immer in der entsprechenden Kommunikationssituation auf.

Um diese Unterscheidung zu automatisieren, soll der Spuranalysator eine Statistik über Fehlerereignisse und Fehlersituationen führen und eine *Fehlerquote* ermitteln. Die Fehlerquote q_c einer Fehlerereignisklasse c ist der Quotient

$$q_c = e_c / s_c$$

aus der Anzahl der Fehlerereignisse e_c dieser Klasse und der absoluten Häufigkeit s_c der zugehörigen Kommunikationssituation. Neben der schon besprochenen Klassifikation der Fehlerereignisse muß dazu außerdem eine Klassifikation der Kommunikationssituationen eingeführt werden. Die genauen Kriterien hierzu sind in Kapitel 3 zu finden.

Schließlich soll die Fehlerquote in ihrem zeitlichen Verlauf korreliert werden mit der beobachteten Kommunikationslast. Dies ist ein einfaches Kriterium, um den Spezialfall lastabhängiger Fehlerklassen heuristisch zu identifizieren: Je höher der ermittelte Korrelationskoeffizient, desto wahrscheinlicher handelt es sich um einen lastabhängigen Fehler.

Die folgende Liste faßt die in diesem Abschnitt erläuterten Funktionsmerkmale des Spuranalysators auf einen Blick zusammen:

1. Fehlerereignisse erkennen
2. Klassen ähnlicher Fehlerereignisse ermitteln
3. PICS-Parameter schätzen
4. Fehlerquoten aus dem Auftretensmuster im Zeitverlauf berechnen
5. Lastabhängige Fehler an der Lastkorrelation erkennen

Besondere Anforderungen

Dieser Abschnitt präzisiert die Aufgabenstellung, indem besondere Anforderungen definiert werden, die die Lösung erfüllen soll. An diesen Anforderungen orientieren sich die späteren Entwurfsentscheidungen wesentlich.

Der Lösungsansatz soll *protokollunabhängig* sein, damit dieselbe Problemstellung nicht für jedes Protokoll neu bearbeitet werden muß, was aufwendig und fehlerträchtig wäre. Es sollen existierende formale Spezifikationsmethoden verwendbar sein, um das korrekte Kommunikationsverhalten für den Analysator zu modellieren.

Die Spuranalyse soll möglichst *in Echtzeit* möglich sein. Diese Fähigkeit hängt natürlich stark mit der Rechenleistung des Systems zusammen, auf dem der Analysealgorithmus jeweils implementiert ist, und auch von der Last auf der beobachteten Kommunikationsstrecke. Dennoch lassen sich aus dem Wunsch nach Echtzeiteinsatz einige Minimalforderungen ableiten:

- Der Speicheraufwand des Verfahrens sollte im Mittel nicht von der Beobachtungsdauer abhängen.
- Der Zeitaufwand des Verfahrens sollte höchstens linear von der Beobachtungsdauer abhängen, die Rechenlast also konstant sein.
- Der PDU-Durchsatz des Verfahrens sollte auf zeitgemäßer Hardware von der Größenordnung her einen Echtzeiteinsatz in praktisch relevanten Fällen möglich erscheinen lassen.

Die ersten beiden Forderungen spiegeln wieder, daß die Ressourcen des Analysators keine Obergrenze für die Beobachtungsdauer setzen sollen. Denn Echtzeitbetrieb ist vor allem bei langen Beobachtungszeiten interessant, für kurze Beobachtungen läßt sich die Spur auch aufzeichnen und anschließend „offline" analysieren. Das entwickelte Verfahren soll sich technisch sowohl für den Betrieb „online", d. h. mit während der Analyse eintreffenden Nachrichten, als auch „offline", d. h. anhand einer als Datei auf Massenspeicher vorliegenden Spur, eignen.

Die Analyse des Protokollverhaltens soll unbedingt auch die Einhaltung vom Protokoll geforderter *Zeitbedingungen* prüfen. Das ist wichtig, weil in praktisch relevanten Protokollen Zeitüberschreitungen eine wichtige Rolle spielen als Auslöser für bestimmte Protokollprozeduren. Nicht eingehaltene oder unter den Kommunikationspartnern inkompatibel eingestellte Zeitfenster sind darüber hinaus eine wahrscheinliche und in der Praxis häufig anzutreffende Fehlerursache.

Die vom Analysator vorgenommene Zeitüberprüfung muß Toleranzintervalle zulassen, da beobachtete Zeiten infolge von Signallaufzeit, Nachrichtendauer, Pufferverzögerung und Meßungenauigkeiten auch bei korrektem Verhalten der IUT von den in der Spezifikation geforderten Zeitbedingungen abweichen können.

Die entscheidende Voraussetzung für die praktische Brauchbarkeit eines Spuranalysators ist, daß dieser seine Analyse *bei jedem beliebigem Zustand der Kommunikation* beginnen kann. Denn es ist praktisch unerwünscht, daß man auf bestimmte Synchronisationsbedingungen im Kommunikationsablauf warten muß, die möglicherweise – wenn überhaupt – sehr selten auftreten oder aber von außen veranlaßt werden müssen, was im Widerspruch zum Konzept der passiven Beobachtung stünde. Der Fall eines unbekannten Protokollzustands tritt außerdem nach *jedem* erkannten Protokollverstoß der IUT auf, denn jede Annahme über deren Folgezustand wäre Spekulation. Dies spiegelt sich auch wieder in der Definition 2 eines Fehlerereignisses. *Ezust* und *Bochmann* schlagen in [EvB95] als eine mögliche Erweiterung ihres Verfahrens vor, allen Zustandsvariablen im Prüfautomaten des Analysators ein Attribut „undefiniert" beizufügen. Der im nächsten Kapitel dargestellte Ansatz dieser Arbeit ist allgemeiner, er enthält die Lösung mit dem Attribut „undefiniert" als einen Spezialfall.

Eine Schwierigkeit bei der passiven Spuranalyse entsteht dadurch, daß die auf einem bidirektionalen Kommunikationskanal gemachten Beobachtungen nicht als vollständig sequentiell angesehen werden können. Figur A.7 stellt den Basisfall dar, der zu einer mehrdeutigen Sequentialisierung der beobachteten Aktionen führt: Der Analysator beobachtet erst eine Eingabe a zur IUT und kurze Zeit später eine Ausgabe b von ihr. Diese beiden PDUs können, müssen aber nicht, sich zwischen dem Beobachtungspunkt und der IUT gekreuzt haben. Das heißt, die beiden entsprechenden Aktionsfolgen $\langle \alpha, \beta \rangle$ und $\langle \beta', \alpha' \rangle$ wären mögliche Sequentialisierungen aus Sicht der IUT, die die Beobachtung erklären.

In [EvB95] wird das Sequentialisierungsproblem ebenfalls erwähnt und durch eine zusätzliche Option gelöst, die Sequenzüberprüfung zwischen bestimmten Nachrichtenwarteschlangen der Spezifikation zu unterdrücken. Im hier vorgestellten Ansatz, der nicht nur die Reihenfolge, sondern auch Zeiten berücksichtigt, sollte eine genauere Prüfung erfolgen: Wenn und nur wenn das Zeitintervall Δt in Figur A.7 unter einer festgelegten oberen Schranke liegt, muß die Aktionsfolge mit Nachrichtenkreuzung als Alternative in Betracht gezogen werden.

Das Analyseverfahren muß sich für *nichtdeterministisch spezifizierte Protokolle* eignen. Dies ist zum einen erforderlich, weil viele Protokolle von vornherein nichtdeterministisch spezifiziert sind. Aber selbst im Falle eines deterministischen Zielprotokolls wird das Protokollverhalten de facto nichtdeterministisch, sobald – wie oben beschrieben – die über die SAPs ausgetauschten Primitiven sich der Beobachtung entziehen, der IUT-Zustand bei Analysebeginn unbekannt ist und mehrdeutige Sequentialisierungen auftreten.

2.2.2 Protokoll-Lernen

Dieser Abschnitt präzisiert die Aufgabe, die Regeln eines Kommunikationsprotokolls aus einem fehlerfreien Beispielverkehr maschinell zu erlernen. Dazu wird im folgenden das Einsatzszenario für einen lernenden Spuranalysator beschrieben. Anschließend werden der Lernvorgang genauer untersucht und strukturiert und vereinfachende Annahmen über die Lernaufgabe eingeführt.

Einsatzszenario

Das physische Einsatzszenario eines lernenden Spuranalysators entspricht dem in Figur A.4 wiedergegebenen für das nichtlernende Verfahren: Der Analysator beobachtet den Kommunikationsverkehr über die Verbindung zwischen System A, der IUT, und seinem Partner B. Logisch teilt sich der Einsatz eines lernenden Spuranalysators in zwei Phasen, eine *Lernphase* und eine *Prüfphase*.

In der Lernphase liest der Lernalgorithmus die PDUs der Schicht n, ohne in die Kommunikation einzugreifen, und leitet aus dieser Beobachtung Regeln ab, die das *korrekte* Protokollverhalten der IUT beschreiben (siehe Figur A.8). Dazu muß vorausgesetzt werden, daß sich sowohl die IUT als auch alle für den Transport der Schicht-n-PDUs verantwortlichen tieferen Protokollschichten tatsächlich protokollkonform verhalten, wie in Abschnitt 2.1.4 ausgeführt; dies deutet das Häkchen in der Figur an.

Man beachte, daß es *nicht* erforderlich ist, daß sich die Partnerinstanz B der IUT korrekt verhält. Wenn sie das tut, treten nie Fehlerprozeduren im Verhalten von A auf, so daß diese

natürlich auch nicht gelernt werden können. Dann würde in der Prüfphase jeder Fehler von B prinzipiell einen Verstoß gegen die erlernten Regeln darstellen. Um also die auftretenden Fehler genauso wie beim nicht lernenden Verfahren sicher einem Kommunikationspartner zuordnen zu können, wäre es sogar wünschenwert, wenn B in der Lernphase auch Verhaltensfehler zeigte. Dies ist jedoch praktisch zu schwer erfüllbar, um grundsätzlich gefordert zu werden.

In der späteren *Prüfphase* kann dann potentiell fehlerbehafteter Kommunikationsverkehr anhand der gelernten Regeln überprüft werden, so daß diese Phase im wesentlichen dem Einsatz des nicht lernenden Analysators nach Abschnitt 2.2.1 entspricht. Allerdings soll die Prüfphase des lernenden Analysators auf eine Protokollschicht, Schicht n, beschränkt sein (siehe Figur A.9).

Annahmen zum Lernvorgang

In diesem Abschnitt wird die Aufgabe des Protokollregellernens durch einige einschränkende Annahmen vereinfacht, um eine Lösung im Rahmen dieser Arbeit zu ermöglichen, und in zwei Teilaufgaben zerlegt.

Die ideale Lernaufgabe im Rahmen der bisherigen Ausführungen würde mit PDUs als Eingabe auskommen, die jeweils als eine Bitkette kodiert und darüber hinaus nur noch mit einem Zeitstempel versehen sind. Die Semantik einer PDU ergibt sich aus den verschiedenen Teilen dieser Bitkette, den *Datenfeldern* der PDU. Folgendes sind die wesentlichen vorkommenden Feldarten:

- *Typinformationen* geben den Typ der PDU oder manchmal auch nur eines Teils von ihr an. Typinformationen entscheiden damit über die Semantik weiterer Teile der PDU.
- *Längeninformationen* geben an, wie viele Bits an irgendeiner Stelle der PDU ein bestimmtes Feld bilden. Solche Informationen haben keinerlei Semantik für das Protokoll, müssen aber ausgewertet werden, um die PDU richtig dekodieren zu können.
- *Daten* sind einfache numerische, textuelle oder andere Informationen, die im Rahmen des Protokolls eine bestimmte Semantik haben, deren konkreter Wert jedoch keine Bedeutung für die Dekodierung anderer Teile der PDU hat.
- *Nutzdaten* sind die Inhalte der Felder bestimmter PDUs, die ohne Semantik für das betrachtete Protokoll im Auftrag der darüberliegenden Protokollschicht $n+1$ übertragen werden. Diese müssen bei der Analyse des Verhaltens der Schicht n vollständig ignoriert werden.

Diese Feldarten erlauben einen derart komplizierten PDU-Aufbau, daß es unmöglich erscheint, die Grenzen, Arten und Bedeutungen der einzelnen Felder vollständig aus den Bitketten der PDUs zu erlernen. Insbesondere die Trennung zwischen Nutzdaten und protokollrelevanten Informationen kann gar nicht gelingen, weil die Nutzdaten einem höheren Protokoll gehorchen, so daß der Lerner zwangsläufig alle Protokollschichten ab n aufwärts zu lernen versuchen würde. Damit wäre eine Mehrschichtenspezifikation zu rekonstruieren, wie sie –

Informationselement	**Typ**	**Länge [Oktette]**
Protokolldiskriminator	Pflicht	1
Call reference	Pflicht	4
Message type	Pflicht	2
Message length	Pflicht	2
AAL parameters	optional	4–21
ATM user traffic descriptor	Pflicht	12–20
Broadband bearer capability	Pflicht	6–7
Broadband high layer information	optional	4–13
Broadband repeat indicator	optional	4–5
Broadband low layer information	optional	4–17
Called party number	optional	≥ 4
Called party subadress	optional	4–25
Calling party number	optional	≥ 4
Calling party subadress	optional	4–25
Connection identifier	optional	4–9
End-to-end transit delay	optional	4–10
Notification indicator	optional	≥ 4
OAM traffic descriptor	optional	4–6
QOS parameter	Pflicht	6
Broadband sending complete	optional	4–5
Transit network selection	optional	≥ 4

Tabelle 2.1: PDU *Setup* aus dem Protokoll Q.2931 [IT94b] als Beispiel für eine komplexe kontextabhängige PDU-Syntax.

wegen der Komplexität und mangelnden Handhabbarkeit – im Protokoll-Engineering gerade nicht eingesetzt wird.

Tabelle 2.1 illustriert die mögliche Komplexität des PDU-Aufbaus beispielhaft anhand der *Informationselemente*, die in einer *SETUP*-PDU des Protokolls Q.2931 [IT94b] enthalten sein können (optional) bzw. müssen (Pflicht) und den Bereichen, in denen sich die Längen der einzelnen Informationselemente bewegen können. Außerdem sind die einzelnen Elemente ebenfalls komplex strukturiert, und ihre Reihenfolge ist weitgehend variabel.

Es ist festzuhalten, daß das Erlernen von Protokollregeln auf Bitkettenebene ein wünschenswertes Fernziel bleibt. In dieser Arbeit, die das Erlernen von Protokollregeln nach Wissen des Autors erstmalig untersucht, soll folgende Vereinfachung angenommen werden:

> Die Dekodierung der PDUs der untersuchten Protokollschicht wird vorausgesetzt. Die PDUs der Trainingsmenge sind gegeben als attributierte Symbole einer endlichen Symbolmenge Σ, wobei jedes Symbol einem PDU-Typ *für eine Kommunikationsrichtung* entspricht. Durch die Unterscheidung in ein- und ausgehende PDUs enthält Σ bei symmetrischen Protokollen also doppelt so viele Elemente, wie es PDU-Typen gibt. Zu jedem $a \in \Sigma$ existiert eine bekannte Attributierung mit $k_a \in \mathbb{N}$ numerischen Attributen. Eine PDU hat also die Form
>
> $$a(x_1; \ldots ; x_{k_a})$$
>
> mit $a \in \Sigma$ und $x_1, \ldots , x_{k_a} \in \mathbb{N}$.

Ferner wird der Regelerwerb nicht in Echtzeit gefordert, sondern wegen des hohen zu erwartenden Rechenaufwands im allgemeinen anhand einer in einer Datei abgelegten Spur durchgeführt. Die Prüfphase dagegen soll, ähnlich wie für die erste Teilaufgabe in Abschnitt 2.2.1 dargestellt, möglichst in Echtzeit erfolgen können, sofern Analysator-Hardware und Einsatzsituation nicht zu ungünstig gewählt sind.

Der Lernvorgang wird in folgende zwei Lernphasen eingeteilt:

1. Erlernen des endlichen Automaten der PDU-Typen, also eines endlichen Automaten über dem Alphabet Σ. Diese Aufgabe entspricht der Identifikation einer regulären Sprache aus positivem Text und ist nach [Gol67] nicht berechenbar (vgl. Abschnitt 2.1.4). Figur A.10 veranschaulicht das Lernen der regulären PDU-Typen-Sprache.

2. Erlernen von Regeln, die gültige Kombinationen der PDU-Attributwerte innerhalb eines Fensters von w aufeinanderfolgenden Nachrichten beschreiben. Die Fenstergröße w ist ein Parameter des Lernverfahrens, der den Lernaufwand und die Genauigkeit des Prüfvorgangs entscheidend beeinflußt. Infolge der festen Attributierung pro PDU-Typ ergibt sich für jede PDU-Typen-Folge $u \in \Sigma^w$ ebenfalls eine feste Attributierung aus der Aneinanderreihung der Attribute der einzelnen PDU-Typen aus u. Formal läßt sich ein Prädikat ok_u definieren, so daß

   $$u^1(x_1^1; \ldots ; x_{k_{u_1}}^1), \ldots , u^w(x_1^w; \ldots ; x_{k_{u_w}}^w) \quad \text{protokollkonform}$$

   g.d.w.

   $$ok_u(x_1^1; \ldots ; x_{k_{u_1}}^1; \ldots ; x_1^w; \ldots ; x_{k_{u_w}}^w)$$

Figur A.11 veranschaulicht das Lernen eines ok-Prädikats für eine beispielhafte Fenstergröße von $w = 3$. Natürlich muß eine solche Attributregel für *jedes* w-Tupel von PDU-Typen gelernt werden, das innerhalb des in Phase 1 gelernten Automaten auftritt.

Weil in beiden Phasen *generalisierendes Lernen aus positiven Beispielen* stattfindet, ist das Lernverfahren vollständig heuristisch. Die Minimalanforderung an das Lernresultat ist, daß es die Trainingmenge als Gutfall akzeptiert. Trotzdem lassen sich weder eine zu starke Generalisierung, die Fehler in der Prüfphase „übersehen" könnte, noch eine zu schwache Generalisierung, die bestimmte nicht trainierte Gutfälle als Fehler einstufen würde, mit Sicherheit ausschließen.

Das für den nichtlernenden Spuranalysator beschriebene Problem der Nachrichtenkreuzung besteht in gleicher Weise auch für das Lernverfahren. Da für letzteres keinerlei Information über die internen Aktionen der IUT vorliegt, muß das Lernverfahren die Fälle mit Nachrichtenkreuzung als eigene Pfade im gelernten Automaten erfassen.

Kapitel 3

Spuranalyse

In diesem Kapitel wird ein neues Verfahren zur Spuranalyse im Online-Betrieb gemäß den in Abschnitt 2.1.3 dargestellten Anforderungen entwickelt. Das hier vorgestellte Verfahren wurde in [Mus97] erstmals veröffentlicht und gründet sich auf Vorarbeiten in [Mus94, Han95]. Es trägt den Namen *FollowSM* (*follow a state machine*, Nachvollziehen eines Automaten).

Das Kapitel beginnt mit einer theoretischen Fundierung der Spuranalyse einschließlich einer Untersuchung zur Berechenbarkeit. Anschließend wird das dem *FollowSM*-Verfahren zugrundeliegende vereinfachte Protokollmodell formal eingeführt und untersucht. Die algorithmische Umsetzung des Analyseverfahrens ist Thema des darauffolgenden Abschnitts. Schließlich wird beschrieben, wie *FollowSM* anhand von vorliegenden SDL-Spezifikationen eingesetzt werden kann.

3.1 Theoretische Fundierung

Dieser Abschnitt beschäftigt sich mit den theoretischen Grundlagen für die zu erarbeitende Lösung, nämlich mit der Wahl des Spezifikationskalküls, mit dessen Hilfe das Sollverhalten beschrieben wird, und der Frage, ob die gestellte Aufgabe im allgemeinsten Fall ein berechenbares Problem darstellt.

3.1.1 Wahl des Spezifikationskalküls

Charakteristisch für den ersten Aufgabenteil ist, daß das „Wissen" um das korrekte Protokollverhalten – kurz *Sollverhalten* – dem Analysator von außen vorgegeben, gewissermaßen eingebaut ist. In Kapitel 2 wurden als mögliche theoretische Konzepte hierzu einerseits die *Prozeßalgebra*, andererseits die *erweiterten endlichen Automaten* vorgestellt.

Aus folgenden Gründen wird für diese Arbeit dem Automatenkonzept der Vorzug eingeräumt:

1. Im Bereich der praktischen Protokollstandardisierung herrscht die automatenbasierte Spezifikationsprache SDL der ITU vor. SDL-Spezifikationen übernehmen in aktuellen ITU-Standards (z. B. [IT94a]) die Rolle der ausschlaggebenden Referenz.

2. Automatenbasierte Spezifikationen erlauben eine einfachere und für menschliche Benutzer leichter verständliche Repräsentation von Kommunikationszuständen. In der Prozeßalgebra wird nichtterminierendes Protokollverhalten durch Verhaltensrekursion modelliert, so daß ein Kommunikationszustand durch Positionen innerhalb der aktiven Prozeßterme und Informationen über die aktuellen Rekursionsebenen und die auf den einzelnen Ebenen gebildeten Variablenbindungen darzustellen wäre. Im Automatenkalkül genügen dagegen pro Automat (EFSM) der Spezifikation der Basiszustand und die Belegung der Zustandsvariablen, eventuell noch die Belegung der Warteschlangen an den Ein- und Ausgängen der einzelnen EFSMs. Diese Zustandsnotation ist intuitiver und übersichtlicher.

3. In der Prozeßalgebra wird im Grundsatz von der Unterscheidung zwischen Eingaben und Ausgaben abstrahiert, stattdessen betrachtet man *Aktionen* mit einer Anzahl von Teilnehmern. Außerdem führt das Paradigma der synchronen Kommunikation dazu, daß der Empfänger einer Nachricht durch seine vorhandene oder nicht vorhandene Bereitschaft zur Kommunikation prinzipiell immer einen Einfluß auf den Sender ausübt, was in der physikalischen Natur realer Kommunikationssysteme keine Entsprechung hat.

Die Auswirkungen von Punkt 3 sollen kurz anhand eines Beispiels verdeutlicht werden. Figur A.12 zeigt links ein Stück einer Protokollspezifikation als LTS, wie es im Prozeßalgebra-Kalkül auftreten kann. Rechts ist eine zugehörige Beobachtung als verzweigungsfreies Verhalten aufgezeichnet. Die Aktion c in der Beobachtung verträgt sich nur mit dem rechten Zweig der Spezifikation, den die IUT – aufgrund der nicht synchronisierten τ-Aktion an seinem Anfang – auf eigene Initiative nichtdeterministisch ausgewählt haben muß. Die zweite beobachtete Aktion a widerspricht daher der Spezifikation. Dabei sind folgende zwei Fälle zu unterscheiden:

1. a ist eine *Ausgabe* der IUT. In diesem Fall liegt mit Sicherheit ein Protokollverstoß der IUT vor.

2. a ist eine *Eingabe* an die IUT. a hätte von der IUT verarbeitet werden können, wenn die entsprechende Nachricht vor der τ-Aktion bei der IUT eingetroffen wäre. Sofern c eine Ausgabe darstellt, die sich mit a zwischen dem Beobachtungpunkt und der Partnerinstanz der IUT kreuzt, kann auch die Partnerinstanz nicht „wissen", daß a nun zur Blockierung führt.

Natürlich lassen sich solche Mehrdeutigkeiten durch Erweiterungen des Kalküls oder besondere Gültigkeitsbedingungen für die Spezifikation, wie beispielsweise in [HT97] vorgeführt, durchaus beseitigen oder entschärfen. Dennoch erscheint das asynchrone Kommunikationsparadigma des Automatenkalküls besser geeignet, gerade mit Fehlersituationen in realen Kommunikationssystemen formal umzugehen.

Daher wird das Automatenkonzept als formale Basis in dieser Arbeit gewählt. Als Sprache für die Vorgabe des Sollverhaltens soll SDL verwendet werden, weil sie die in der Praxis verbreiteteste automatenbasierte Protokollspezifikationssprache ist.

3.1.2 Berechenbarkeit

In diesem Abschnitt wird gezeigt, daß bereits die Spuranalyse einer IUT, für deren Spezifikation eine einzige EFSM dient, nicht berechenbar ist. Dies gilt damit erst recht für Spezifikationen im vollen Sprachumfang von SDL.

Zunächst wird dazu die *vereinfachte Spuranalyse* definiert, deren Unberechenbarkeit zu zeigen ist. Für den Beweis wird ein Spezialfall der *vereinfachten Spuranalyse* auf eine Turing-Maschine (TM) übertragen und die von dieser berechnete Funktion auf das Allgemeine Halteproblem reduziert, dessen Unberechenbarkeit hinlänglich bekannt ist (z. B. [Kai72]).

Definition 3 Vereinfachte Spuranalyse: *Gegeben seien die EFSM-Spezifikation A einer Instanz, die über einen unbeobachteten Kanal s und einen beobachteten Kanal p kommuniziert, und ein beobachteter Kommunikationsablauf o über p. Die vereinfachte Spuranalyse ermittelt, ob ein Kommunikationsablauf i über s und eine spezifikationskonforme Transitionsfolge t^* von A existieren, so daß o ein zulässiges Verhalten von A kennzeichnet.*

Figur A.13 illustriert die gegebene Definition.

Nun soll folgender *Spezialfall* betrachtet werden: Der Automat A startet in seinem Initialzustand, erhält dann eine vollständige Eingabe i über s und gibt eine resultierende Ausgabe o über p aus. A berechnet also eine Funktion $f_A : I^* \rightarrow O^*$, wobei I das Eingabealphabet an s und O das Ausgabealphabet an p bezeichnen. Damit läßt sich die vereinfachte Spuranalyse einer Beobachtung o leicht formalisieren als die Entscheidung der Formel

$$Y_A(o) \quad \equiv \quad \exists i \in I^*(f_A(i) = o)$$

In dieser Umgebung kann nun jede Turing-Maschine m als Automat A verwendet werden, beispielsweise, indem das Band der Turing-Maschine als eine Datenvariable der EFSM über Zeichenketten modelliert wird und die Zustände von m direkt als Zustände von A verwendet werden[1]. Mit diesem Wissen läßt sich der betrachtete Spezialfall der vereinfachten Spuranalyse wie folgt als Funktion $Y : \mathbb{N} \times \mathbb{N} \rightarrow \{0; 1\}$ beschreiben. Dabei muß die Beschreibung der beobachteten Instanz zum Funktionsargument werden:

$$Y(m, o) = \begin{cases} 1 & \text{wenn} \quad \exists i(m(i) = o) \\ 0 & \textit{sonst} \end{cases}$$

wobei m die Gödelnummer einer Turing-Maschine ist, die für alle Eingaben hält, und $m(i)$ ist die Ausgabe der durch m kodierten Turing-Maschine bei Eingabe i. Über die Funktionswerte von Y für Eingaben m, die für gewisse i nicht halten, wird keine Aussage gemacht. Diese Einschränkung ist sinnvoll, weil m ja die Spezifikation der beobachteten Instanz repräsentiert – sie wird als verklemmungsfrei vorausgesetzt. Ein Spuranalysealgorithmus könnte also den Instanzautomaten mit gewissen Eingaben „probeweise ausführen", ohne die eigene Termination zu gefährden. Der folgende Beweis zeigt, daß ein Algorithmus für die vereinfachte Spuranalyse selbst unter dieser günstigen Annahme nicht existiert.

[1] Dies erfordert ein EFSM-Modell, das auch leere Ein- und Ausgaben zuläßt, damit die Berechnung selbst nicht beobachtet wird. Für EFSMs mit mehreren Kanälen ist diese Sichtweise natürlich, da Transitionen hier ohnehin nur mit Interaktionen *auf einem Teil* der Kanäle verknüpft sind.

Theorem 1 *Es gibt keine berechenbare Funktion* $Y : \mathbb{N} \times \mathbb{N} \rightarrow \{0; 1\}$*, die der gegebenen Beschreibung entspricht.*

Beweis: Angenommen, es gibt eine solche berechenbare Funktion Y, also eine Turing-Maschine, die sie berechnet.

Es gibt bekanntlich ([Kai72]) eine *universelle Turing-Maschine* U, die die Gödelnummer einer Turing-Maschine und eine Eingabe erwartet und die angegebene Maschine auf dieser Eingabe simuliert. Daraus läßt sich eine Turing-Maschine S konstruieren, bei der $S(m, i, n) = 1$ g.d.w. die TM mit Gödelnummer m bei Eingabe i innerhalb von n Schritten terminiert, sonst $S(m, i, n) = 0$. Dies ist z. B. möglich durch die Erweiterung von U um einen Schrittzähler, der mit einem berechenbarkeitstheoretisch unerheblichen zweiten Band realisiert wird.

Bezeichne nun $S_{m,i}$ die Gödelnummer der Turing-Maschine, bei der die Argumente m und i gegenüber S fixiert sind, so daß $S_{m,i}(n) = S(m, i, n)$. Diese Gödelnummer ist aus der Gödelnummer von S, m und i berechenbar, weil die Gödelnummer einer TM, die ein festes m und i auf das Band schreibt, berechenbar ist und die Operation der Verkettung dieser TM mit S ebenfalls auf arithmetische Operationen zurückführbar ist. Man kann also eine Maschine H konstruieren, so daß:

$$H(m, i) \stackrel{def}{=} Y(S_{m,i}, 1)$$

H berechnet also erst die Gödelnummer $S_{m,i}$, schreibt eine 1 als zweite Eingabe aufs Band und startet abschließend Y. H berechnet:

> Gibt es eine Eingabe n an $S_{m,i}$, so daß 1 als Ausgabe erzeugt wird (nach Definition von Y)?

Und damit:

> Gibt es eine Schrittzahl n, nach der die Turing-Maschine mit Gödelnummer m hält, wenn i eingegeben wird?

Dies ist offenbar das Allgemeine Halteproblem, welches unberechenbar ist. Dieser Widerspruch macht die Annahme der Existenz von Y unhaltbar. □

Weil bereits der untersuchte Spezialfall der *vereinfachten* Spuranalyse nicht berechenbar ist, gilt diese Feststellung auch für alle allgemeineren Fälle.

3.2 Lösungsansatz

Nachdem im vorangegangenen Abschnitt gezeigt werden konnte, daß die Aufgabe der Spuranalyse – selbst ohne Einbeziehung von Zeitbedingungen – nicht uneingeschränkt automatisierbar ist, führt dieser Abschnitt ein reduziertes Protokollmodell ein, das an die Anforderungen hinsichtlich Berechenbarkeit und Effizienz der Spuranalyse angepaßt ist und auf dem das *FollowSM*-Verfahren basiert. Dieses Modell enthält auch einen exakten Formalismus zur Beschreibung von Zeitbedingungen für das Protokollverhalten. Der hier dargestellte Ansatz wurde bereits in [Mus97] publiziert.

Die Konsequenzen, die sich aus der Aufgabenstellung für dieses Protokollmodell ergeben, werden sich als so gravierend erweisen, daß ein direkte Rückgriff auf eine geeignete Untermenge von SDL zur Beschreibung des Sollverhaltens verworfen werden mußte. Der stattdessen eingeschlagene Lösungsweg besteht darin,

1. den in diesem Abschnitt eingeführte Hilfsformalismus zur formal exakten Beschreibung eines Sollverhaltens zur Verfügung zu stellen, und
2. anschließend ein Werkzeug zu entwerfen, das SDL-Spezifikationen in diesen Hilfsformalismus überführt. Dieser Schritt ist wegen der gezeigten Unberechenbarkeit mit einem Verlust an Exaktheit verbunden und erfordert möglicherweise manuelle Hilfestellungen, da Entwurfentscheidungen und Kompromisse festzulegen sind.

Der zweite Schritt ist Gegenstand eines späteren Abschnitts in diesem Kapitel.

3.2.1 Protokollmodellierung und einfache Spuranalyse

Die *einfache Spuranalyse* im Sinne des *FollowSM*-Verfahrens basiert auf einer Protokollspezifikation in Form eines einzelnen erweiterten endlichen Automaten (EFSM). Dieses Modell wird im folgenden als *beobachtbarer zeitbehafteter erweiterter endlicher Automat* (*observable timed extended finite state machine*, OTEFSM) bezeichnet. Die Beschränkung auf einen *einzelnen* Automaten stellt im Hinblick auf die Modellierungsmächtigkeit keine prinzipielle Einschränkung dar, da *erweiterte* endliche Automaten – wie schon in Abschnitt 3.1.2 angesprochen – Turing-Maschinen simulieren können, sofern Zustandsvariablen mit unendlichem Wertebereich verwendbar sind.

Beobachtbare zeitbehaftete EFSM

Ein beobachtbarer zeitbehafteter erweiterter endlicher Automat (OTEFSM) M ist ein Quadrupel $M = (Q, T, \Sigma, \Delta, d)$ mit folgenden Eigenschaften:

- Q ist eine Menge von *einfachen Zuständen*, die denjenigen Anteil der Zustandsinformation der Protokollmaschine beschreiben, der *nicht zeitbezogen* ist. Q darf unendlich sein.
- T ist eine endliche und möglicherweise leere Menge von *Zeitgebernamen*.
- Σ ist eine möglicherweise unendliche Menge von *Ein- oder Ausgabesymbolen*.
- Δ ist eine endliche Menge von *Transitionen* (Zustandsübergängen). Diese Transitionen sind weiter strukturiert, wie weiter unten erläutert wird.
- $d \in \mathbb{N}$ ist die *maximale Ausgabeverzögerung*, diejenige Zeitspanne, die höchstens verstreichen kann, bis eine von einer Transition erzeugte Ausgabe tatsächlich sichtbar wird (vgl. Sequentialisierungsproblem in Abschnitt 2.2.1).

Wie das letzte Element andeutet, wird die Zeit im OTEFSM-Modell als natürliche Zahl repräsentiert. Diese Entscheidung ist allerdings willkürlich, benötigt wird nur die totale Ordnung, nicht die Abzählbarkeit der Zeitpunkte.

Im folgenden wird Indizierung mit Namen benutzt, um Komponentenbeziehungen klarzustellen, wo immer das erforderlich ist. Beispielweise bezeichnet Q_M die Zustandsraumkomponente der OTEFSM M. Indizierung mit natürlichen Zahlen kommt darüber hinaus zum Einsatz, um Projektion auszudrücken, z. B. in $p_2 = 4$ für $p = (3, 4)$.

Ausgehend von den einfachen Zuständen Q und Zeitgebernamen T erhält man die Menge der *vollständigen Zustände* S der OTEFSM formal als

$$S \stackrel{def}{=} Q \times (T \to \mathbb{N}_\infty^2)$$

mit Elementen $(q, \tau) \in S$. Dabei bezeichnet $\mathbb{N}_\infty$ die Menge der natürlichen Zahlen, erweitert um einen Wert „unendlich". Die praktische Bedeutung ist, daß ein vollständiger Zustand (q, τ) aus einem einfachen Zustand q, einer Variablenbelegung, und einer Zuordnung von Zeitintervallen aus $\mathbb{N}_\infty^2$ zu den Zeitgebernamen, einer Zeitgeberbelegung, besteht. Die Zeitintervalle geben den absoluten Zeitraum an, innerhalb dessen der jeweilige Zeitgeber ablaufen wird. Damit lassen sich Nichtdeterminismus und Meßungenauigkeiten im Zusammenhang mit zeitbedingtem Verhalten modellieren. Das Paar (∞, ∞) steht für einen inaktiven Zeitgeber, der „nie" ablaufen wird.

Auf Zeitgeberzuständen $\tau, \tau' \in T \to \mathbb{N}_\infty^2$ bezeichne $\leq$ eine partielle Ordnung, die den Intervalleinschluß *für alle Zeitgeber* repräsentiert:

$$\tau \leq \tau' \text{ g.d.w. } \forall \psi \in T\,([\tau(\psi)_1, \tau(\psi)_2] \subseteq [\tau'(\psi)_1, \tau'(\psi)_2])$$

Schließlich wird noch ein spezieller Wert $\perp \notin T \cup \Sigma$ verwendet werden, der Nichtexistenz ausdrückt. In diesem Zusammenhang werden die Abkürzungen T_0 und Σ_0 eingeführt:

$$\begin{aligned} T_0 &\stackrel{def}{=} T \cup \{\perp\} \\ \Sigma_0 &\stackrel{def}{=} \Sigma \cup \{\perp\} \end{aligned}$$

Jede Transition $\delta \in \Delta$ ist ein Quintupel $\delta = (\psi, e, \alpha, \theta, \pi)$:

- $\psi \in T_0$ bestimmt denjenigen Zeitgeber, der ablaufen muß, damit δ zum Schalten aktiviert wird. Im Fall $\psi = \perp$ handelt es sich um eine nicht zeitabhängige Transition.
- $e \in \mathbb{P}(Q \times \Sigma \times \Sigma_0) \cup \mathbb{P}(Q \times \{\perp\} \times \Sigma_0)$ ist das *Aktivierungsprädikat* für die Transition δ. e definiert sowohl die Zustandsbedingung, unter der δ schalten kann oder muß, als auch ihr Ein-/Ausgabeverhalten. Die erste Potenzmenge in der Vereinigung oben gilt für Transitionen, die durch Eingaben ausgelöst werden, während eingabelose Transitionen von der zweiten abgedeckt werden. Im übrigen dürfen ausschließlich solche eingabelosen Transitionen zeitabhängig sein.
- $\alpha : e \to Q$ ist die *Zustandsüberführungsfunktion*, die die Änderung des einfachen Zustandes der OTEFSM beschreibt, die eintritt, sobald δ schaltet.

- $\theta : T \to \mathbb{N}^2 \cup \{(\infty, \infty)\}$ ist eine partielle Funktion, die den Effekt von δ auf den Zeitgeberzustand beschreibt. $\theta(\psi) = (a, b)$ bedeutet, daß der Zeitgeber ψ beim Schalten von δ gestartet wird und frühestens nach a, spätestens nach b Zeiteinheiten ablaufen wird. Der Sonderfall $a = b = \infty$ wird verwendet, um das Anhalten eines Zeitgebers zu spezifizieren.
- $\pi \in \mathbb{N}$ ist die *Priorität* von δ. Größere Zahlen sollen hier für höhere Prioritäten stehen.

Semantik

Nachdem die Struktur einer OTEFSM beschrieben wurde, erfolgt in diesem Abschnitt die Definition der Semantik des Protokollmodells im Hinblick auf zulässige Beobachtungen und auf die – einfache – Spuranalyse gegenüber einer OTEFSM-Spezifikation.

Als erster Schritt hierzu wird die Funktion $\Psi : \Delta \times \mathbb{N} \times S \times \Sigma_0 \to \mathbb{N}^2$ eingeführt, die eine *Aktivierungspriorität* ausdrückt. $\Psi(\delta, t, s, i)$ ist die Aktivierungspriorität der Transition δ zum Zeitpunkt t angesichts des vollständigen OTEFSM-Zustands s und einer anstehenden Eingabe i, die von δ zu verarbeiten ist; $i = \perp$, falls keine Eingabe stattfindet. Diese Aktivierungspriorität wird durch ein *Paar* von Zahlen ausgedrückt, weil sie sowohl von der Art der Transition, dem Automatenzustand und der Zeit abhängt als auch von der angegebenen Transitionspriorität π_δ, und zwar in dieser Reihenfolge. Die totale Ordnung auf diesen zweistufigen Prioritäten ist daher gegeben durch

$$(a, b) < (c, d) \quad \text{g.d.w.} \quad a < c \vee (a = c \wedge b < d)$$

Die Definition von Ψ unterscheidet zwischen folgenden, z. T. dynamisch bestimmten Transitionstypen:

spontan $\Psi(\delta, t, s, i) = (1, 0)$ wenn $\psi_\delta = \perp \wedge \pi_\delta = 0 \wedge \exists o \in \Sigma_0((s, \perp, o) \in e_\delta)$

unmittelbar $\Psi(\delta, t, s, i) = (3, \pi_\delta)$ wenn $\psi_\delta = \perp \wedge \pi_\delta > 0 \wedge \exists o \in \Sigma_0((s, \perp, o) \in e_\delta)$

eingabeabhängig $\Psi(\delta, t, s, i) = (2, \pi_\delta)$ wenn $\psi_\delta = \perp \wedge \exists o \in \Sigma_0((s, i, o) \in e_\delta)$

zeitabhängig $\Psi(\delta, t, s, i) = (1, \pi_\delta)$ wenn $\psi_\delta \neq \perp \wedge \tau_s(\psi_\delta)_1 \leq t \wedge \exists o \in \Sigma_0((s, \perp, o) \in e_\delta)$

inaktiv $\Psi(\delta, t, s, i) = (0, 0)$ sonst.

Eine weitere Funktion, $\Phi : \Delta \times \mathbb{N} \times S \to \{0; 1\}$, markiert *verspätete* Transitionen δ, d. h. aktivierte zeitabhängige Transitionen, deren Zeitgeber im Zustand s zur Zeit t bereits abgelaufen ist. Es gilt:

$$\Phi(\delta, t, s) = \begin{cases} 1 & \text{wenn } \psi_\delta \neq \perp \wedge t \geq \tau_s(\psi_\delta)_2 \wedge \exists o \in \Sigma_0((s, \perp, o) \in e_\delta) \\ 0 & \text{sonst} \end{cases} \tag{3.1}$$

Es folgt die Definition der *lokalen Zustandsübergangsrelation*

$$\lambda \subseteq S \times \Delta \times \mathbb{N}^2 \times \Sigma_0^2 \times S.$$

Diese beschreibt den Effekt des Schaltens einer Transition innerhalb eines festgelegten Zeitraums auf den vollständigen OTEFSM-Zustand.

$$((q,\tau),\delta,t_1,t_2,i,o,(q',\tau')) \in \lambda$$

bedeutet: Zustand (q,τ) wird überführt in Zustand (q',τ'), wenn die Transition δ zu einem Zeitpunkt zwischen t_1 und t_2 schaltet und dabei das Symbol i liest und das Symbol o ausgibt (wiederum $i = \perp$ bzw. $o = \perp$ für den Wegfall der Kommunikation). Mit einer intuitiven Pfeilnotation lautet die Definition:

$$\begin{aligned}
&(q,\tau) \xrightarrow[i,o]{\delta,t_1,t_2} (q',\tau') \quad \text{g.d.w.} \\
&\quad \forall t \in [t_1,t_2](\Psi(\delta,t,(q,\tau),i) > (0,0)) \wedge q' = \alpha_\delta((q,\tau),i,o) \\
&\wedge \quad \forall x \in T : \tau'(x) = \begin{cases} (t_1+\theta_\delta(x)_1, t_2+\theta_\delta(x)_2) & \text{wenn } x \in \mathrm{dom}(\theta_\delta) \\ (\infty,\infty) & \text{wenn } x = \psi_\delta \wedge x \notin \mathrm{dom}(\theta_\delta) \\ \tau(x) & \text{sonst} \end{cases} \qquad (3.2)
\end{aligned}$$

Der zweite Fall bei der Änderung des Zeitgeberzustands modelliert, daß ein Zeitgeber automatisch deaktiviert wird, nachdem er eine Transition ausgelöst hat.

Nun kann die *globale Zustandsübergangsrelation* λ^* formuliert werden, die λ so erweitert, daß die durch Ψ festgelegte Priorisierung Beachtung findet. Ihre Funktionalität ist:

$$\lambda^* \subseteq S \times \mathbb{N}^2 \times \Delta \times (\Sigma_0 \times \mathbb{N})^2 \times S \times \mathbb{N}^2$$

Hierbei steht

$$(s,t_1,t_2,\delta,i,t_i,o,t_o,s') \in \lambda^*$$

für den Sachverhalt, daß ausgehend von Zustand s, der zu irgendeinem Zeitpunkt zwischen t_1 und t_2 betreten wurde, als nächster Schaltvorgang einer Transition das Schalten von δ zu irgendeinem Zeitpunkt zwischen t_1' und t_2' in Frage kommt, mit dem resultierenden Folgezustand s', wobei die Eingabe i mit ihrem Zeitstempel t_i gelesen und die Ausgabe o mit ihrem Zeitstempel t_o erzeugt werden. Wiederum, $i = \perp$ bzw. $o = \perp$ stehen für keine beobachtbare Ein- bzw. Ausgabe; in diesem Fall sind die Zeitstempel bedeutungslos. Mit der Abkürzung $P = \Psi(\delta,t_2',s,i)$ wird die Relation λ^* – in Pfeilnotation – definiert durch:

$$\begin{aligned}
&s,t_1,t_2 \xrightarrow[(i,t_i),(o,t_o)]{\delta} s',t_1',t_2' \quad \text{g.d.w.} \quad s \xrightarrow[i,o]{\delta,t_1',t_2'} s' \\
&\quad \wedge \quad t_1 \le t_2 \wedge t_1 \le t_1' \le t_2' \\
&\quad \wedge \quad o \neq \perp \Rightarrow t_2' \le t_o \le t_1' + d \wedge i \neq \perp \Rightarrow t_1' = t_2' = t_i \\
&\quad \wedge \quad \forall \delta^* \in \Delta\,((\Psi(\delta^*,t_2',s,i) \le P \vee (\psi_{\delta^*} \neq \psi_\delta \wedge \Psi(\delta^*,t_2',s,i) < (2,0))) \\
&\qquad \wedge \forall t^* \in [t_1,t_2'[\ (\Psi(\delta^*,t^*,s,\perp) < (2,0) \wedge \Phi(\delta^*,t^*,s) = 0)) \qquad (3.3)
\end{aligned}$$

Vom Standpunkt der Ausführung einer OTEFSM-Spezifikation drückt dies alles aus, daß Transitionen entweder ausgelöst werden können durch ihre zustandsabhängige Aktivierungsbedingung allein (*spontane* und *unmittelbare* Transitionen) oder durch die Ankunft einer Eingabe-PDU (*eingabeabhängige* Transitionen) oder durch den Ablauf eines Zeitgebers (*zeitabhängige* und *verspätete* Transitionen). *Spontane* Transitionen haben eine „Kannschalten"-Semantik, d. h., ihre Aktivierung erzwingt keinen Schaltvorgang. *Unmittelbare*

Transitionen dagegen *müssen* schalten, sobald sie aktiviert sind, und unterdrücken dann alle anderen Transitionstypen. Aktivierte *eingabeabhängige* Transitionen weisen auch eine „Muß-schalten"-Semantik auf, können jedoch durch *unmittelbare* Transitionen zurückgestellt werden. *Zeitabhängige* Transitionen sind ebenfalls als vom „Muß-schalten"-Typ einzustufen, wobei der tatsächliche Zeitgeberablaufzeitpunkt zufällig gezogen wird innerhalb des Ablaufintervalls $[t_1, t_2]$ des zugehörigen Zeitgebers. Wenn das Aktivierungsprädikat einer solchen Transition wahr wird, *nachdem* ihr Zeitgeber bereits abgelaufen ist, so wird von einer *verspäteten* Transition gesprochen, die zwar *für den Zeitpunkt dieses Ereignisses* keine anderen Transitionen überdeckt, jedoch vor allen Alternativen schalten muß.

Vom Standpunkt der Spuranalyse ist das beschriebene Verhalten der zeitabhängigen Transitionen äquivalent zu den obigen Ausführungen zu modellieren als eine „Kann-schalten"-Situation innerhalb des Zeitgeberablaufintervalls und eine „Muß-schalten"-Regel ab dessen Obergrenze t_2, weil der tatsächliche Ablaufzeitpunkt im Rahmen der Analyse ja nicht von vornherein bekannt ist.

Zwischen aktivierten Transitionen gleichen Typs erfolgt die Priorisierung gemäß π_δ. Sofern dennoch mehr als eine Transition aktiv bleibt, findet eine nichtdeterministische Auswahl statt. Der Schaltzeitpunkt einer eingabeabhängigen Transition wird immer mit dem Zeitstempel der zugehörigen Eingabe-PDU identifiziert. Ausgaben können dagegen bis zu d_δ Zeiteinheiten nach ihrer Erzeugung beobachtet werden, Ausgabezeitstempel stellen also keine festen Schaltzeitpunkte dar.

Eine Beobachtung ist genau dann eine zulässige Spur im Rahmen des Protokolls, wenn ein Anfangszustand $s^0 \in S$ existiert und eine Folge global zulässiger Transitionen der Form

$$s^0, 0, 0 \xrightarrow[(i^1,t_i^1),(o^1,t_o^1)]{\delta^1} s^1, t_1^1, t_2^1 \xrightarrow[(i^2,t_i^2),(o^2,t_o^2)]{\delta^2} \ldots \xrightarrow[(i^l,t_i^l),(o^l,t_o^l)]{\delta^l} s^l, t_1^l, t_2^l,$$

die genau die beobachteten Ein- und Ausgaben mit den korrekten Zeitstempeln verarbeitet bzw. erzeugt – $\perp$s wie üblich ignoriert für den Fall eingabe- bzw. ausgabeloser Transitionen.

Die – einfache – Spuranalyse gegenüber einer OTEFSM-Spezifikation bedeutet die Entscheidung der Zulässigkeit einer gegebenen Spur.

Damit ist die Definition der Semantik des OTEFSM-Modells abgeschlossen.

Berechenbarkeit

Die Definition der OTEFSM ist an zahlreichen Stellen gerade so zugeschnitten, daß die Spuranalyse gegen eine OTEFSM-Spezifikation effizient berechenbar ist. Das folgende Lemma bringt die wichtigste dieser Eigenschaften zum Ausdruck.

Lemma 1 *Betrachte einen Zustandsübergang*

$$(q, \tau), t_1, t_2 \xrightarrow[(i,t_i),(o,t_o)]{\delta} (q', \tau'), t_1', t_2'$$

gemäß λ^* und einen vollständigen OTEFSM-Zustand $(q, \hat{\tau}), \hat{t}_1, \hat{t}_2$ mit Eintrittsintervall, so daß

$$\hat{\tau} \geq \tau \wedge [\hat{t}_1, \hat{t}_2] \supseteq [t_1, t_2].$$

Dann gilt:

a) Es existiert ein Schaltzeitintervall $[\hat{t}'_1, \hat{t}'_2] \supseteq [t'_1, t'_2]$ *mit*

$$(q, \hat{\tau}), \hat{t}_1, \hat{t}_2 \xrightarrow[(i,t_i),(o,t_o)]{\delta} (q', \hat{\tau}'), \hat{t}'_1, \hat{t}'_2.$$

b) Für jede λ^**-Transition entsprechend* **a** *gilt* $\tau' \leq \hat{\tau}'$.

Beweis: Teil **a**. Es wird gezeigt, daß **a** für

$$[\hat{t}'_1, \hat{t}'_2] \stackrel{def}{=} [t'_1, t'_2]$$

erfüllt ist. Nach (3.3) muß zunächst

$$(q, \hat{\tau}) \xrightarrow[i,o]{\delta, t'_1, t'_2} (q', \hat{\tau}') \tag{3.4}$$

erfüllt sein. An der Definition von Φ erkennt man, daß

$$\begin{aligned} \hat{\tau} \geq \tau \quad &\Rightarrow \quad \Phi(\delta, t, (q, \hat{\tau}), i) = \Phi(\delta, t, (q, \tau), i) \\ &\vee \quad (\Phi(\delta, t, (q, \hat{\tau}), i) = (1, \pi_\delta) \wedge \Phi(\delta, t, (q, \tau), i) = (0, 0)) \end{aligned} \tag{3.5}$$

und damit auf jeden Fall $\Phi(\delta, t, (q, \hat{\tau}), i) \geq \Phi(\delta, t, (q, \tau), i)$. Hieraus folgt die Bedingung $\Phi(\delta, t, (q, \hat{\tau}), i) > (0, 0)$ aus (3.2). q' hängt nicht von $\hat{\tau}$ ab, somit gilt (3.4).

Die Bedingungen für t_1, t_2 in (3.3) sind für $\hat{t}_1, \hat{t}_2$ auch erfüllt wegen $\hat{t}_1 \leq t_1 \leq t_2 \leq \hat{t}_2$.

Nun zur Disjunktion in der vorletzten Zeile von (3.3): Mit $\hat{P} \stackrel{def}{=} \Phi(\delta, t'_2, (q, \hat{\tau}), i)$ gilt $\Phi(\delta^*, t, (q, \tau), i) \leq P \leq \hat{P}$ (vgl. (3.5)). Problematisch ist also einzig der Fall

$$\Phi(\delta^*, t, (q, \hat{\tau}), i) = (1, \pi_{\delta^*}) \wedge \Phi(\delta^*, t, (q, \tau), i) = (0, 0).$$

Dann ist aber für $\psi_{\delta^*} = \psi_\delta$ wegen (3.2) stets $\Phi(\delta^*, t, (q, \hat{\tau}), i) = \hat{P}$ und somit die linke Seite der Disjunktion gültig, für $\psi_{\delta^*} \neq \psi_\delta$ ergibt sich $\Phi(\delta^*, t, (q, \hat{\tau}), i) = (1, \pi_{\delta^*}) < (2, 0)$.

Bleibt die letzte Zeile von (3.3) zu betrachten: Die Ungleichung links bleibt wegen (3.5) auch für $\hat{\tau}$ erfüllt. Aus $\Phi(\delta^*, t^*, (q, \tau)) = 0$ folgt nach (3.1):

$$t < \tau(\psi_\delta)_2 \leq \hat{\tau}(\psi_\delta)_2,$$

also $\Phi((\delta^*, t^*, (q, \hat{\tau})) = 0$. Damit ist **a** gezeigt.

b ist eine direkte Folgerung aus der Berechnungsvorschrift für die resultierenden Zeitgeberzustände τ' und $\hat{\tau}'$ in (3.2): Wegen $\hat{t}_1 \leq t_1 \wedge \hat{t}_2 \geq t_2$ folgt

$$[\hat{t}_1 + \theta_\delta(x)_1, \hat{t}_2 + \theta_\delta(x)_2] \supseteq [t_1 + \theta_\delta(x)_1, t_2 + \theta_\delta(x)_2]$$

und damit **b**. □

Mit diesem Lemma läßt sich nun der entscheidende Satz dieses Abschnitts, nämlich die Berechenbarkeit der einfachen Spuranalyse gegenüber einer OTEFSM-Spezifikation, beweisen.

Theorem 2 *Die einfache Spuranalyse gegenüber einer OTEFSM-Spezifikation ist berechenbar, sofern der Ausgangszustand s^0 bekannt ist und die Spezifikation keine unendlich langen Transitionssequenzen ohne Ein- und Ausgaben erlaubt.*

Bemerkung: Da beobachtete Ein-/Ausgabefolgen naturgemäß stets endlich sind, können die in der Spuranalyse betrachteten Zeiten t für mögliche Schaltzeitpunkte von Transitionen stets anhand dem Ende t_E der Beobachtungsdauer nach oben beschränkt werden, $t < t_E$. Der folgende Beweis macht von dieser Schranke Gebrauch.

Beweis: Zu einer zulässigen Beobachtung (I, O) mit Beobachtungsende t_E existiert eine erklärende Aktionsfolge A gemäß (3.3). Es ist zu zeigen, daß diese Existenz algorithmisch entscheidbar ist. Es wird gezeigt, daß zu jedem solchen A eine eindeutig bestimmte *normalisierte* Aktionsfolge A_n existiert, die (I, O) erklärt, und daß die Menge aller zur Beobachtung (I, O) passenden normalisierten Aktionsfolgen endlich und berechenbar ist. Folglich ist die einfache Spuranalyse berechenbar durch die erschöpfende Suche nach einer solchen normalisierten Aktionsfolge A_n.

Sei A gegeben, sei ferner

$$(q, \tau), t_1^0, t_2^0 \xrightarrow[(i,t_i),(o,t_o)]{\delta} (q', \tau'), t_1', t_2'$$

einer der Zustandsübergänge in A (vgl. (3.3)). Es gilt

$$\exists \hat{t}_1, \hat{t}_2 \in \mathbb{N} \left(\hat{t}_1 \leq \hat{t}_2 \leq t_E \quad \wedge \right.$$
$$\left. [\hat{t}_1, \hat{t}_2] = \bigcup \left\{ [t_1, t_2] \middle| (q, \tau), t_1^0, t_2^0 \xrightarrow[(i,t_i),(o,t_o)]{\delta} (q', \hat{\tau}'), t_1, t_2 \right\} \right), \tag{3.6}$$

in Worten: Zu jedem Zustandsübergang existiert ein größtes umhüllendes Schaltzeitintervall, das mit der zugehörigen Beobachtung und dem resultierenden *einfachen* Zustand q' verträglich ist. Dies folgt daraus, daß in (3.3)

a) die für t_1' und t_2' geltenden Schranken über die Intervallbedingung $t_1' \leq t_2'$ hinaus nicht von der Wahl der jeweils anderen Zeit abhängen,

b) außer $t_1' \leq t_2'$ für t_1' nur untere, für t_2' nur obere Schranken existieren und

c) alle Schranken nach **b** $\leq$- bzw. $\geq$-Bedingungen sind und damit abgeschlossene Intervalle bilden.

Die Forderung $(q, \tau) \rightarrow (q', \tau')$ in (3.3) erfüllt **a**, **b** und **c**, weil Ψ monoton steigend in t ist und (3.2) daher höchstens eine kleinste untere Schranke für t_1' liefert. t_1' kommt ansonsten nur noch im Zusammenhang mit den unteren Schranken t_i und $t_o - d_\delta$ in (3.3) vor; auch dies ist mit **a**, **b**, **c** konform. t_2' ist nach Voraussetzung durch t_E sowie ggf. durch t_o bzw. t_i einschließlich nach oben beschränkt; der Fall $t_1' = t_2' = t_i$ für eingabeabhängige Transitionen liefert auch keine neue untere Schranke für t_2' außer t_1'. Die letzte Zeile von (3.3) kann nur einschließliche obere Schranken liefern in Form der Sprungstellen von Φ, da Φ nicht in Abhängigkeit von t^* den Wert $(2, 0)$ erreichen oder überschreiten kann. In der vorletzten Zeile von (3.3) liegt im Fall

$P \geq (2,0)$ entsprechend keinerlei Zeitabhängigkeit vor. Für $P < (2,0)$ ist im Fall $\psi_{\delta'} \neq \psi_\delta$ die Zeit t'_2 ebenfalls unerheblich, weil jede andere Transition mit Priorität von $(2,0)$ oder mehr das Schalten von δ ohnehin ausschließt. Für $\psi_{\delta'} = \psi_\delta$ schließlich sind die Sprungstellen von Ψ für die verglichenen Transitionen jeweils identisch, also spielt t'_2 für diese Zeile in keinem Fall eine Rolle. Dies beschließt den Nachweis von **a**, **b**, **c** und mithin von (3.6).

Das Intervall $[\hat{t}_1, \hat{t}_2]$ ist aufgrund seiner Eigenschaften *eindeutig bestimmt*. Aufgrund von Lemma 1 kann eine normalisierte Aktionsfolge A_n gebildet werden, indem diese größten Schaltzeitintervalle nach (3.6) iterativ von links nach rechts für die ursprünglichen Intervalle in A substituiert werden, denn Lemma 1 garantiert die Zulässigkeit der Schaltschritte trotz der erforderlichen „Verbreiterung" der Zeitgeberablaufintervalle in den τ^k. A_n ist

1. anhand von A eindeutig bestimmt,
2. stellt eine gültige λ^*-Expansion dar und
3. erklärt ebenfalls die Beobachtung (I, O).

Da die Zwischenzustände (q^k, τ^k) durch die Funktionen α_δ und θ_δ deterministisch bestimmt sind[2], T und die Zahl der beobachteten Symbole endlich sind und die Anzahl der Transitionen pro Symbol nach Voraussetzung endlich ist, sind alle normalisierten Aktionsfolgen, die (I, O) erklären könnten, aufzählbar mit Hilfe einer erschöpfenden Suche im Zustandsraum der OTEFSM. Eine erklärende Aktionsfolge A existiert genau dann, wenn eine erklärende normalisierte Aktionsfolge A_n gefunden wird. □

Dieses Ergebnis kann nun verglichen werden mit dem Unberechenbarkeitsresultat von Abschnitt 3.1.2. Die hier eingeführte Voraussetzung, daß keine unendlich langen unbeobachtbaren Transitionssequenzen auftreten, entspricht oben der Annahme, daß die Protokollmaschine A für jede beliebige Eingabe über den unbeobachteten Kanal hält. Zeiten wurden in 3.1.2 überhaupt nicht betrachtet. Der entscheidende Unterschied besteht folglich darin, daß die OTEFSM *ohne unbeobachteten Eingabekanal* modelliert ist, die Projektion der Protokollmaschine auf ihr beobachtetes Verhalten also Teil des Entwurfs der OTEFSM-Spezifikation des Sollverhaltens ist. Wenn diese Projektion eines Protokolls P nicht ohne Verfälschung des Sollverhaltens erfolgen kann, ist die vollständige und korrekte Spuranalyse von P daher nicht berechenbar.

3.2.2 Unsicherheitsbehaftete Spuranalyse

Die wesentliche Schwierigkeit, die die einfache Spuranalyse eines OTEFSM-Modells noch nicht löst, ist die Einschränkung, daß der Anfangszustand s_0 bekannt sein muß (vgl. Abschnitt 2.2.1). Der entscheidende Schritt, der das *FollowSM*-Analyseverfahren von dieser Einschränkung befreit, besteht darin, bei der Spuranalyse *unsicherheitsbehaftete Zustandsinformationen* verarbeiten zu können. Dies bedeutet, daß der Suchalgorithmus im Zustandsraum so verallgemeinert wird, daß er nicht mehr auf einzelnen Werten von Zustandvariablen operiert, sondern auf Teilmengen ihrer Wertebereiche. Dieser Abschnitt erläutert die Erweiterung der

[2]Man beachte, daß die Ausgabe einer Transition δ dagegen nicht, wie in [EvB95], deterministisch bestimmt zu sein braucht, weil e_δ nur eine Relation ist.

einfachen Spuranalyse auf die Verarbeitung von Unsicherheit. Das resultierende Konzept heißt *unsicherheitsbehaftete Spuranalyse* und stellt eine Verallgemeinerung der bisher behandelten OTEFSM-Spuranalyse dar, die aus diesem Grund zur besseren Unterscheidung als *einfache Spuranalyse* bezeichnet wurde.

Überblick

Für einen informellen Überblick betrachte man einen Zustandsraum mit einem Variablenpaar

$$(a, b) \in \{0, 1, .., 255\}^2.$$

Die Beschreibung des Anfangszustands lautet

$$s^0 \in (0..255, 0..255),$$

eine genauere Aussage läßt sich ohne Kenntnis der Zustands- und Kommunikationshistorie nicht machen.

Sei nun δ eine Kandidaten-Transition aus einer OTEFSM-Spezifikation. Angenommen, ihr Aktivierungsprädikat e_δ definiert eine zugehörige Ausgabenachricht $\sigma(x)$ mit $x \in \{0, .., 255\}$ und fordert $a < x$, und die Zustandsüberführungsfunktion α_δ bewirkt $a \leftarrow a + 10$; $b \leftarrow 17$. Die nächste beobachtete Ausgabenachricht sei $\sigma(42)$. Kein Spuranalyseverfahren, das auf der direkten Ausführung einer Referenzimplementierung basiert, kann mit diesen Informationen irgend etwas anfangen.

Das *FollowSM*-Verfahren dagegen kann mit der Unsicherheit umgehen, die das geschilderte Szenario läßt: Es instantiiert das Aktivierungsprädikat anhand der konkreten Eingabenachricht zu $a < 42$, schränkt dann die vorliegende Beschreibung des Zustands s^0 auf diejenige Teilmenge ein, für die δ aktiviert ist, und wendet δ schließlich spekulativ auf diese intermediäre Zustandsbeschreibung an:

$$s^0 \in (0..255, 0..255) \xrightarrow{a<42} s' \in (0..41, 0..255) \xrightarrow{a \leftarrow a+10; b \leftarrow 17} s^1 \in (10..51, 17)$$

s^1 ist – nachdem lediglich eine einzige Transition betrachtet wurde – bereits eine viel weniger unsichere Beschreibung des Protokollzustands, als es s^0 gewesen ist. Natürlich müssen alle anderen Transitionen der Spezifikation ebenfalls für eine spekulative Ausführung in Betracht gezogen werden. Viele davon werden normalerweise in dem – nichts ausschließenden – Zustand s_0 ebenfalls aktiviert sein. Dies gilt insbesondere für *sämtliche* Transitionen, die nicht mit Kommunikationsprimitiven verknüpft sind.

Allerdings kann diese Form der unsicherheitsbehafteten Zustandsexpansion nicht in einer *vollständigen* und *korrekten* Art und Weise durchgeführt werden, d. h. mit dem Resultat *„pass"* genau dann, wenn eine zulässige Aktionsfolge zu einer Beobachtung existiert. Dies liegt an Speicherplatz- und Zeitbeschränkungen im Zusammenhang mit der Tatsache, daß das Erfüllbarkeitsproblem für allgemeine boolesche Ausdrücke (SAT-Problem) nicht effizient berechenbar ist [HS78]. Aus diesem Grund erfolgt eine approximative Lösung des unsicherheitsbehafteten Spuranalyseproblems, wobei die folgenden beiden Bedingungen eingehalten werden:

1. Keine zulässige Spur darf mit dem Urteil „fail" zurückgewiesen werden. Das heißt, der Algorithmus muß im Hinblick auf die Zurückweisung von Spuren wenigstens *korrekt* sein, wenn auch nicht unbedingt *vollständig*.

2. Sobald die Zustandsraumsuche einen „sicheren" Zwischenzustand erreicht hat, soll der auf diesen Zeitpunkt folgende Teil der Analyse sowohl *korrekt* als auch *vollständig* sein, d. h. alle späteren Protokollverstöße mit Sicherheit finden.

Formale Definition

Für die formale Beschreibung der unsicherheitsbehafteten Spuranalyse muß zunächst ein *Darstellungraum* R für den einfachen Zustandsraum Q der jeweiligen OTEFSM-Spezifikation ausgewählt werden. Dieser Darstellungsraum ist die Menge der darstellbaren Teilmengen von Q. Er muß als Minimalanforderung erfüllen, daß wenigstens der vollständig unsichere Zustand auf der einen Seite und sämtliche vollständig sicheren Zustände auf der anderen Seite darstellbar sind:

$$R \subseteq \mathbb{P}Q \setminus \emptyset \quad \wedge \quad Q \in R \wedge \forall q \in Q(\{q\} \in R) \tag{3.7}$$

Die Menge der *vollständigen unsicherheitsbehafteten Zustandsbeschreibungen* wird nun mit U bezeichnet:

$$U = R \times (T \to \mathbb{N}_\infty^2)$$

Für die weitere Beschreibung soll die Hilfsfunktion $\mathrm{st} : U \to \mathbb{P}S$ dazu dienen, um zu einer gegebenen unsicherheitsbehafteten Zustandsbeschreibung $u = (r, \tau)$ die Menge aller sicheren vollständigen Zustände zu liefern, die u einschließt, nämlich

$$\mathrm{st}(r, \tau) \stackrel{def}{=} \{(q, \tau) | q \in r\}.$$

Die *unsicherheitsbehaftete Spuranalyse* kann nun beschrieben werden als eine Erweiterung von λ^* zu einer Relation μ^*, bei der U anstelle von S eingesetzt wird. Sie hat die Signatur

$$\mu^* \subseteq U \times \mathbb{N}^2 \times \Delta \times (\Sigma_0 \times \mathbb{N})^2 \times U \times \mathbb{N}^2.$$

Mit der Abkürzung

$$v' \stackrel{def}{=} \left\{ s' \in S \,\middle|\, \exists s \in \mathrm{st}(u) : s, t_1, t_2 \xrightarrow[(i,t_i),(o,t_o)]{\delta} s', t_1', t_2' \right\}$$

für diejenigen Zustände, die von irgendeinem in u enthaltenen Zustand direkt erreichbar sind, lautet die Definition von μ^* in Pfeilnotation:

$$\begin{aligned} u, t_1, t_2 \xrightarrow[(i,t_i),(o,t_o)]{\delta} u', t_1', t_2' \quad & \text{g.d.w.} \\ & t_1 \le t_2 \wedge t_1 \le t_1' \le t_2' \wedge t_2' \le t_o \le t_1' + d \wedge i \neq \perp \Rightarrow t_1' = t_2' = t_i \\ \wedge \quad & \mathrm{st}(u') \supseteq v' \wedge (|\mathrm{st}(u)| = 1 \Rightarrow \mathrm{st}(u') = v') \end{aligned} \tag{3.8}$$

In der unsicherheitsbehafteten Spuranalyse gilt eine Beobachtung genau dann als zulässige Spur des spezifizierten Protokolls, wenn eine nach (3.8) zulässige Folge von Zustandsübergängen der Form

$$u^0, 0, 0 \xrightarrow[(i^1,t_i^1),(o^1,t_o^1)]{\delta^1} u^1, t_1^1, t_2^1 \xrightarrow[(i^2,t_i^2),(o^2,t_o^2)]{\delta^2} \ldots \xrightarrow[(i^l,t_i^l),(o^l,t_o^l)]{\delta^l} u^l, t_1^l, t_2^l$$

existiert, die genau die beobachteten Symbole mit den zutreffenden Zeitstempeln liest bzw. erzeugt, mit $u^0 = (Q, T \times \{(0, \infty)\})$ als initiale Zustandsbeschreibung, die für die vollständige Ungewißheit über den tatsächlichen Zustand beim Beobachtungsbeginn steht.

Eigenschaften

Die in Abschnitt 3.2.2 gestellten Anforderungen 1 und 2 an die unsicherheitsbehaftete Spuranalyse hinsichtlich Vollständigkeit und Korrektheit werden nun in den Theoremen 3 und 4 gezeigt. Theorem 3 befaßt sich mit der in Anforderung 1 ausgedrückten Korrektheitsbedingung.

Theorem 3 *Jede Beobachtung, die im Sinne der einfachen OTEFSM-Spuranalyse zulässig ist, ist auch zulässig im Sinne der unsicherheitsbehafteten OTEFSM-Spuranalyse.*

Beweis: Es genügt zu zeigen, daß für jede λ^*-Expansion ein bei u^0 beginnender μ^*-Expansionspfad existiert, der zu der gleichen Beobachtung führt. Der Beweis erfolgt durch vollständige Induktion über die Länge der Aktionsfolge.

Als Induktionsannahme betrachte man einen Zustand $s = (q, \tau) \in S$ und eine diesen Zustand enthaltene unsicherheitsbehaftete Zustandsbeschreibung $u = (r, \tau_u) \in U$, es gilt also $q \in r \wedge \tau \leq \tau_u$. Sei ferner

$$(q, \tau), t_1, t_2 \xrightarrow[(i,t_i),(o,t_o)]{\delta} (q', \tau'), t_1', t_2'$$

ein gemäß λ^* (3.3) gültiger Zustandsübergang.

Aus Lemma 1 folgt, daß ein $\tau_u' \geq \tau'$ existiert, für das auch der Übergang

$$(q, \tau_u), t_1, t_2 \xrightarrow[(i,t_i),(o,t_o)]{\delta} (q', \tau_u'), t_1', t_2'$$

gültig ist. Nach Definition von μ^* existiert folglich ein gültiger unsicherheitsbehafteter Zustandsübergang

$$(r, \tau_u), t_1, t_2 \xrightarrow[(i,t_i),(o,t_o)]{\delta} (r', \tau_u'), t_1', t_2'$$

mit $q' \in r'$, womit der Induktionsschritt abgeschlossen ist.

Weil das festgelegte u^0 für die unsicherheitsbehaftete und jedes beliebige s^0 für die einfache Spuranalyse die Induktionsannahme erfüllen, folgt die Behauptung durch vollständige Induktion. □

Theorem 4 drückt Anforderung 2 aus, die Vollständigkeit der unsicherheitsbehafteten Spuranalyse nach einem sicheren Zwischenzustand. Unter Hinzunahme von Anforderung 1 ist dies gleichbedeutend mit der Äquivalenz von einfacher und unsicherheitsbehafteter Spuranalyse für die Zeit nach dem sicheren Zwischenzustand.

Theorem 4 Einfache *und* unsicherheitsbehaftete *Spuranalyse sind äquivalent, wenn eine unsicherheitsfreie Beschreibung des Initialzustands der Form* $s^0 = (q^0, \tau^0)$ *bzw.* $u^0 = (\{q^0\}, \tau^0)$ *gegeben ist.*

Beweis: Wie ein Vergleich der Definitionen von λ^* und μ^* zeigt, genügt es nachzuweisen, daß sichere Zustandsbeschreibungen von μ^* stets wieder in sichere Zustandsbeschreibungen expandiert werden. Formal ausgedrückt ist zu zeigen, daß $|v'| = 1$ immer gilt, wenn $|\,\mathrm{st}(u)| = 1$. In diesem Fall sind μ^* und λ^* offensichtlich äquivalent.

$|v'| = 1$ gilt stets im Falle sicherer Ausgangszustände, weil der resultierende Zustand s eines Expansionsschrittes durch α_δ, ϕ_δ und λ eindeutig bestimmt ist, sofern die schaltende Transition δ, das Schaltzeitintervall und die beobachteten Interaktionen feststehen. □

Abschließend soll noch die Berechenbarkeit der unsicherheitsbehafteten Spuranalyse gezeigt werden.

Theorem 5 *Die unsicherheitsbehaftete Spuranalyse gegenüber einer OTEFSM-Spezifikation ist berechenbar, sofern die Spezifikation keine unendlich langen Transitionssequenzen ohne Ein- und Ausgaben erlaubt.*

Beweisskizze: Dieser Beweis läuft sehr ähnlich wie derjenige des Theorems 2. Hier wird eine normalisierte Aktionsfolge gebildet, indem alle einfachen Zustände r^k in den Zwischenschritten auf $r^k \stackrel{def}{=} Q \in R$ festgelegt und wiederum iterativ von links nach rechts die maximalen Schaltzeitintervalle substituiert werden, wie sie sich aus den Zeitgeberzuständen τ^k, den Eintrittsintervallen $[t_1^k, t_2^k]$ und den Zeitstempeln t_i^k, t_o^k der ausgetauschten Symbole gemäß (3.3) ergeben. Nach Definition von μ^* ist die so erhaltene Folge A_n zulässig. Sie ist eindeutig bestimmt, weil insbesondere alle vorkommenden Zeitgeberzustände eindeutig bestimmt sind. Sie erklärt gleichfalls die Beobachtung wegen Teil **b** von Lemma 1. Also ist die Berechnung der unsicherheitsbehafteten Spuranalyse möglich durch die erschöpfende Suche nach einer solchen normalisierten Aktionsfolge, die die Beobachtung erklärt. □

3.3 Algorithmische Umsetzung

Dieser Abschnitt beschreibt, wie die zuvor definierte unsicherheitsbehaftete Spuranalyse im *FollowSM*-Verfahren in einem praktisch nutzbaren Analysealgorithmus realisiert worden ist. Dazu wird erst auf die Konsequenzen aus Theorem 5 hinsichtlich der Konvergenz der Zustandsbeschreibung und auf die operationale Behandlung der Unsicherheit eingegangen. Es folgt eine Erläuterung des Kernalgorithmus für die Zustandsexpansion. Danach wird dargelegt, welche Suchstrategie für die Suche im Zustandsraum zu bevorzugen ist, und wie man aus dem Suchbaum eine lineare Zustandsentwicklung erhält. Ein weiterer Unterabschnitt beschreibt die verschiedenen Arten von Analysen, die aus dem Ergebnis der Spuranalyse generiert werden können. Letzter Punkt ist eine Beschreibung des Datenmodells, das der Algorithmus verwendet.

3.3.1 Behandlung der Unsicherheit

Im Beweis von Theorem 5 wird deutlich, daß die Spezifikation der unsicherheitsbehafteten Spuranalyse sehr schwach ist: Als Beschreibung der *einfachen* Zustandsanteile q^k genügen im allgemeinen Obermengen der tatsächlich gemäß der einfachen Spuranalyse möglichen Zustände, so daß sich der Beweis mit Q als „Joker-Zustand" aushelfen kann. Die Spezifikation (3.8) erzwingt lediglich die Einhaltung der Zeitbedingungen und die korrekte Fortschreibung der Zeitgeberzustände τ^k. Dank dieses Freiraums für Implementierungen der unsicherheitsbehafteten Spuranalyse kann der Darstellungsraum R auch für sämtliche Zwischenschritte bei der Zustandsentwicklung verwendet werden, solange die Obermengenbedingung Beachtung findet. Dadurch können auch komplexe Bedingungen für den aktuellen Zustand, die sich im Laufe der Suche ergeben können, zum Preis des Verzichts auf Vollständigkeit (vgl. Abschnitt 3.2.2) effizient behandelt werden. Allein die Wahl eines eingeschränkten Repräsentationsraums R für die Zustandsmengen schließt die Vollständigkeit schon deshalb aus, weil die exakte Menge der möglichen Folgezustände nach einer Transition – je nach Art dieser Transition – möglicherweise nicht repräsentierbar ist.

Jede Implementierung der Spuranalyse muß bei jedem Entwicklungsschritt, der von einer unsicherheitsbehafteten Zustandsbeschreibung ausgeht, einen Kompromiß irgendwo zwischen den exakten Folgezuständen und der unbestimmten Zustandsbeschreibung Q eingehen, einen Kompromiß zwischen Genauigkeit und Effizienz. Ob und wie schnell die Beschreibung des aktuellen Zustands zu einem sicheren Zustand konvergiert, hängt davon ab, wie die Implementierung diesen Spielraum ausnutzt. Die Konvergenz kann jedenfalls nicht aus der Definition der unsicherheitsbehafteten Spuranalyse abgeleitet werden, sondern nur aus dem konkreten Analysatorentwurf für ein konkretes Protokoll.

Datentypen zur Modellierung der Unsicherheit

Beim *FollowSM*-Verfahren wird der Darstellungsraum R nicht quasi elementweise entworfen, sondern ergibt sich, indem für jede einzelne Zustandsvariable der Protokoll-EFSM einschließlich des Basiszustands ein besonderer Datentyp eingesetzt wird, der Teilmengen des Wertebereichs dieser Zustandsvariable gemäß der Minimalforderung (3.7) darstellen kann und die nötigen Prädikate und Operatoren zur Verfügung stellt. Mit n Zustandsvariablen

$$v_1 : Q_1, ..., v_n : Q_n$$

und den zugehörigen die Variablen der Unsicherheit modellierenden Datentypen

$$u_1 : R_1 \in \mathbb{P}Q_1, ..., u_n : R_n \in \mathbb{P}Q_n$$

ergeben sich der Zustand $q \in Q$ und die Darstellung $r \in R$ zu

$$\begin{aligned} q = (v_1, ..., v_n) &: \quad Q = Q_1 \times ... \times Q_n \\ r = (u_1, ..., u_n) &: \quad R = R_1 \times ... \times R_n \end{aligned}$$

mit der Besonderheit, daß dieser Darstellungsraum keinerlei *Abhängigkeiten* zwischen den Werten verschiedener Zustandsvariablen zu modellieren erlaubt – die Zustandsbeschreibung

$$q = (1, 2) \vee q = (2, 1)$$

kann nur ungenau dargestellt werden als

$$r = (1...2, 1...2).$$

Um verschiedene Anforderungen erfüllen zu können, werden für die üblichen Grunddatentypen jeweils mehrere Repräsentationsdatentypen bereitgehalten, die sich in der Darstellungsgenauigkeit unterscheiden. Dazu ein Beispiel: Eine Zustandsvariable mit Wertebereich $0, 1, \ldots, 255$ (ein *Oktett*) läßt sich etwa einerseits darstellen durch eine ganzzahlige Größe, die entweder einen bestimmten Wert der Zustandsvariable annimmt oder einen ausgezeichneten Wert (vielleicht -1) für den Fall *unbekannt*, d. h. den vollständigen Wertebereich. Dies ist genau die Minimallösung gemäß (3.7). Andererseits könnte man auch ein Paar (a, b) von ganzen Zahlen verwenden, die ein Intervall $[a, b]$ beschreiben. Einzelwerte $x \in 0, 1, \ldots, 255$ werden dann durch $a = b = x$ dargestellt, die vollständige Unsicherheit durch $(0, 255)$. Der Extremfall in Richtung maximaler Genauigkeit wäre eine vollständige Teilmengenrepräsentation, etwa durch einen 256-stelligen Bitvektor. Während die ersten beiden Fälle geeignet sind, beispielsweise Sequenznummern mit großem Wertebereich platzsparend darzustellen, bietet sich die dritte Variante z. B. an, um den für Protokollprozeduren entscheidenden Basiszustand, für den es nur eine überschaubare Zahl von Möglichkeiten gibt, ohne Genauigkeitsverlust zu modellieren.

Tabelle 3.1 stellt die beim *FollowSM*-Prototyp verfügbaren Datentypen für Zustandsvariablen dar. In der Spalte *Name, Parameter* sind der jeweilige Datentyp benannt und ggf. seine Parameter spezifiziert, von denen der Wertebereich der modellierten Zustandsvariable abhängt. Die Spalte *Wertebereich* gibt diesen Wertebereich Q der Zustandsvariablen an. In der Spalte *Darstellung* stehen die Variablen, die der jeweilige Datentyp intern zur Darstellung verwendet und die damit einen Rückschluß auf den Speicherplatzbedarf einer entsprechend modellierten Zustandsvariable zulassen. In der Spalte *Semantik* schließlich ist vermerkt, wie sich die repräsentierte Zustandsteilmenge aus der Darstellung ergibt.

Mit diesen fünf Datentypen sind die wichtigsten skalaren Zustandsgrößen in Kommunikationsprotokollen abgedeckt. Die Datentypen *SET* und *LSET* unterscheiden sich nicht funktional, *LSET* bietet lediglich eine effizientere Implementierung für Fälle, in denen der repräsentierende Bitvektor in einer einzelnen ganzzahligen Variable abgelegt werden kann (32 Bits).

Die Zustandsvariablen mit einem Wertebereich „Intervall der ganzen Zahlen" der Form $\{L; ...; U\}$ werden als Ringintervall aufgefaßt, was im Fall $L = 0$ der Restklassenarithmetik modulo $U + 1$ entspricht. Figur A.14 verdeutlicht diese Eigenschaft für das Beispiel $L = 10, U = 21$. Für Protokolle wird die Restklassenarithmetik häufig benötigt, da Sequenznummern typischerweise in dieser Form interpretiert werden (z. B. im SSCOP-Protokoll [IT94a]). Formal ist die Arithmetik auf den Ringintervallen folgendermaßen definiert:

$$\begin{aligned}
a <_{L,U} b \quad &\text{g.d.w.} \quad 0 < b - a \leq \frac{U - L + 1}{2} \vee a - b \geq \frac{U - L + 1}{2} \\
a >_{L,U} b \quad &\text{g.d.w.} \quad 0 < a - b < \frac{U - L + 1}{2} \vee b - a > \frac{U - L + 1}{2} \\
a +_{L,U} b \quad &= \quad L + ((a + b - L) \bmod (U - L + 1)) \\
a -_{L,U} b \quad &= \quad L + ((a - b - L) \bmod (U - L + 1)) \\
&a, b \in \{L, ..., U\}; \quad L, U \in \mathbb{Z}
\end{aligned}$$

Während die Definition der Addition und Subtraktion unmittelbar einsichtig sind, mußte für die Vergleiche – als einzig mögliche Festlegung ohne Kenntnis der jeweiligen Semantik der Werte – eine Links- bzw. Rechts-von-Regel im Sinne eines Uhrzeigers verwendet werden. Diese Interpretation liefert zumindest dann das erwartete Ergebnis, wenn die verglichenen Werte relativ zur Intervallgröße „nahe beieinander" liegen – nämlich bis hin zur halben Intervallgröße.

Operationen und Prädikate

Die jeweilige Protokollspezifikation enthält Prädikate und Operationen auf den Zustandsvariablen des Protokollautomaten. Diese Prädikate und Operationen müssen beim *FollowSM*-Verfahren auf Objekte der hier vorgestellten Datentypen angewandt werden. Dies führt, wie in Abschnitt 3.2.2 beispielhaft ausgeführt, zu Änderungen, insbesondere zu Einschränkungen der Wertemenge, die aktuell für die Belegung einer Zustandsvariable noch in Frage kommt. Die Operationen auf unsicherheitsbehafteten Zustandsvariablen sind damit erheblich komplexer als die auf den Zustandsvariablen in der Protokollspezifikation, die zu jedem Zeitpunkt ja mit genau einen einzelnen (*sicheren*) Wert aus ihrem Wertebereich belegt sind. Der entscheidende Unterschied ist, daß auch die Auswertung eines *Prädikats* P auf einer unsicherheitsbehafteten Zustandsbeschreibung $r \in R$ zu einer neuen Zustandsbeschreibung r' führt, nämlich zu derjenigen Teilmenge von r, für deren Elemente das fragliche Prädikat erfüllt ist. Man kann also ein *unsicherheitsbezogenes* Prädikat P_u aus P ableiten als eine Funktion auf der unsicherheitsbehafteten Zustandsbeschreibung:

$$r' = P_u(r) \supseteq \{q \in r | P(q)\}$$

Die Obermengen-Bedingung ist wieder eine Folge der eingeschränkten Repräsentierbarkeit. Das Ergebnis „falsch" eines konventionellen Prädikats ist äquivalent zu $r' = \emptyset$. Nachdem ein Prädikat dergestalt ausgewertet wurde und „wahr" ergeben hat, d. h. eine *nichtleere* resultierende Zustandsbeschreibung r', müssen alle nachfolgenden Operationen, die gemäß der Protokollspezifikation von der Gültigkeit der mit P modellierten Bedingung abhängen, auf r' operieren, denn sie sind nur dann *möglicherweise* zulässig, *wenn* der *tatsächliche* aktuelle Zustand innerhalb der Zustandsmenge r' liegt.

Im folgenden werden die wesentlichen Operationen auf den Unsicherheit modellierenden Datentypen aufgelistet. Das Symbol Q steht dabei immer für den Zustandsvariablen-Wertebereich und das Symbol R für den Darstellungsraum des jeweiligen Datentypen. Für die angegebenen Spezifikationen wird aus Vereinfachungsgründen

$$R_0 \stackrel{def}{=} R \cup \emptyset$$

verwendet, um mit Hilfe der leeren Wertemenge $\emptyset$ die Nichterfüllung eines Prädikats ausdrücken zu können. Man beachte, daß die leere Menge mit den Datentypen an sich nicht repräsentierbar sein muß, weil sie einem „unmöglichen" Zustand entspricht und ein solcher in keinem Fall weiterverarbeitet zu werden braucht. Der min-Operator auf einer Menge von Mengen bezeichnet in der folgenden Auflistung jeweils eine Menge mit der kleinsten Kardinalität.

- Test auf Gleichheit mit einem Wert b

  Operation: $\mathrm{eq} : R \times Q \to R_0$
  Betrifft: BOOL, INT, RANGE, SET, LSET
  Spezifikation: $\mathrm{eq}(a, b) = \{b\}$

- Test auf Gleichheit zweier Zustandsvariablen a, b

  Operation: $\mathrm{eq} : R \times R \to R_0 \times R_0$
  Betrifft: BOOL, INT, RANGE, SET, LSET
  Spezifikation: $\mathrm{eq}(a, b) = (a', b')$
  $a' = b' = \min\{r \in R | r \supseteq a \cap b\}$

- Test auf „wahr"

  Operation: $\mathrm{tr} : R \to R_0$
  Betrifft: BOOL
  Spezifikation: $\mathrm{tr}(a) = a \cap \mathrm{W}$

- Test auf „falsch"

  Operation: $\mathrm{fl} : R \to R_0$
  Betrifft: BOOL
  Spezifikation: $\mathrm{tr}(a) = a \cap \mathrm{F}$

- Test auf „kleiner als" Wert b

  Operation: $\mathrm{lt} : R \times Q \to R_0$
  Betrifft: INT, RANGE, SET, LSET
  Spezifikation: $\mathrm{lt}(a, b) = \min\{r \in R | r \supseteq \{x \in a | x < b\}\}$

- Kleiner-als-Relation zweier Zustandsvariablen a, b

  Operation: $\mathrm{lt} : R \times R \to R_0 \times R_0$
  Betrifft: INT, RANGE, SET, LSET
  Spezifikation: $\mathrm{lt}(a, b) = (a', b')$
  $a' = \min\{r \in R | r \supseteq \{x \in a | \exists y \in b \bullet x < y\}\}$
  $b' = \min\{r \in R | r \supseteq \{y \in b | \exists x \in a \bullet x < y\}\}$

- Test auf „kleiner oder gleich" Wert b

  Operation: $\mathrm{le} : R \times Q \to R_0$
  Betrifft: INT, RANGE, SET, LSET
  Spezifikation: $\mathrm{le}(a, b) = \min\{r \in R | r \supseteq \{x \in a | x \leq b\}\}$

- Kleiner-oder-gleich-Relation zweier Zustandsvariablen a, b

Operation: $\text{le} : R \times R \to R_0 \times R_0$
Betrifft: INT, RANGE, SET, LSET
Spezifikation: $\text{le}(a, b) = (a', b')$
$a' = \min\{r \in R | r \supseteq \{x \in a | \exists y \in b \bullet x \leq y\}\}$
$b' = \min\{r \in R | r \supseteq \{y \in b | \exists x \in a \bullet x \leq y\}\}$

- Test auf „größer als" Wert b

Operation: $\text{gt} : R \times Q \to R_0$
Betrifft: INT, RANGE, SET, LSET
Spezifikation: $\text{gt}(a, b) = \min\{r \in R | r \supseteq \{x \in a | x > b\}\}$

- Größer-als-Relation zweier Zustandsvariablen a, b

Operation: $\text{gt} : R \times R \to R_0 \times R_0$
Betrifft: INT, RANGE, SET, LSET
Spezifikation: $\text{gt}(a, b) = (a', b')$
$a' = \min\{r \in R | r \supseteq \{x \in a | \exists y \in b \bullet x > y\}\}$
$b' = \min\{r \in R | r \supseteq \{y \in b | \exists x \in a \bullet x > y\}\}$

- Test auf „größer oder gleich" Wert b

Operation: $\text{ge} : R \times Q \to R_0$
Betrifft: INT, RANGE, SET, LSET
Spezifikation: $\text{ge}(a, b) = \min\{r \in R | r \supseteq \{x \in a | x \geq b\}\}$

- Größer-oder-gleich-Relation zweier Zustandsvariablen a, b

Operation: $\text{ge} : R \times R \to R \times R_0$
Betrifft: INT, RANGE, SET, LSET
Spezifikation: $\text{ge}(a, b) = (a', b')$
$a' = \min\{r \in R | r \supseteq \{x \in a | \exists y \in b \bullet x \geq y\}\}$
$b' = \min\{r \in R | r \supseteq \{y \in b | \exists x \in a \bullet x \geq y\}\}$

- Addition eines Wertes b

Operation: $+ : R \times Q \to R$
Betrifft: INT, RANGE, SET, LSET
Spezifikation: $a + b = \min\{r \in R | r \supseteq \{z \in Q | \exists x \in a \bullet z = x + b\}\}$

- Addition zweier Zustandsvariablen a, b

Operation: $+ : R \times R \to R$
Betrifft: INT, RANGE, SET, LSET
Spezifikation: $a + b = \min\{r \in R | r \supseteq \{z \in Q | \exists x \in a, y \in b \bullet z = x + y\}\}$

- Subtraktion eines Wertes b

  Operation: $+ : R \times Q \rightarrow R$
  Betrifft: INT, RANGE, SET, LSET
  Spezifikation: $a - b = a + (-b)$

- Subtraktion zweier Zustandsvariablen a, b

  Operation: $- : R \times R \rightarrow R$
  Betrifft: INT, RANGE, SET, LSET
  Spezifikation: $a - b = \min\{r \in R | r \supseteq \{z \in Q | \exists x \in a, y \in b \bullet z = x - y\}\}$

- Mengenrelation zweier Zustandsvariablen a, b

  Operation: $\&\& : R \times R \rightarrow \{\text{id}, \text{sub}, \text{sup}, \text{ov}, \text{dis}\}$
  Betrifft: BOOL, INT, RANGE, SET, LSET
  Spezifikation:
  $$a\&\&b = \begin{cases} \text{id} & \text{wenn } a = b \\ \text{sub} & \text{wenn } a \subset b \\ \text{sup} & \text{wenn } a \supset b \\ \text{ov} & \text{wenn } a \cap b \neq \emptyset \wedge a \not\subseteq b \wedge a \not\supseteq b \\ \text{dis} & \text{wenn } a \cap b = \emptyset \end{cases}$$

  Die Abkürzungen stehen für *identical* (identisch), *subset* (echte Teilmenge), *superset* (echte Obermenge), *overlapping* (überlappend) und *disjoint* (disjunkt).

- Zustandsvereinigung zweier Zustandsvariablen a, b

  Operation: $| : R \times R \rightarrow R$
  Betrifft: BOOL, INT, RANGE, SET, LSET
  Spezifikation: $a|b = \min\{r \in R | r \supseteq a \cup b\}$

Es ist zu beachten, daß für die Datentypen *RANGE*, *SET* und *LSET* die Operatoren $<, >, +, -$ im Sinne der weiter oben spezifizierten Ringarithmetik $<_{L,U}, >_{L,U}, +_{L,U}, -_{L,U}$ zu interpretieren sind.

Eine nur auf den ersten Blick irritierende Konsequenz aus der Auswertung von Prädikaten unter Unsicherheit ist die Tatsache, daß das logische Gesetz

$$a \wedge \overline{a} = \text{F}$$

nicht mehr gilt: Angenommen, $a = \{17, ..., 42\} \in$ RANGE wird mit den Prädikaten $a \leq 30$ und $a > 30$ untersucht. In beiden Fällen lautet das Ergebnis „wahr", mit den resultierenden eingeschränkten Zustandsbeschreibungen $a = \{17, ..., 30\}$ im ersten und $a = \{31, ..., 42\}$ im zweiten Fall. Anschaulich bedeutet dies, daß angesichts der gegebenen Zustandsbeschreibung in der Protokoll-*Implementierung* jedes der beiden Prädikate als „wahr" ausgewertet werden *könnte*. Daher müssen für das nachfolgende Protokollverhalten auch beide Alternativen, der „ja"- *und* der „nein"-Zweig der Fallunterscheidung, gleichberechtigt als in Frage kommendes korrektes Verhalten in Betracht gezogen werden.

Logische Ausdrücke

Ebenfalls komplizierter als in der konventionellen Auswertung von Bedingungen zeigt sich die Behandlung von Konjunktionen und Disjunktionen mehrerer Einzelprädikate. Zur Diskussion dieses Sachverhalts wird zunächst folgende Schreibweise eingeführt für die Anwendung eines Prädikats P auf eine Zustandsbeschreibung r:

$$P : R \to R_0 \qquad P(r) = r'$$

wobei diejenige bzw. diejenigen Zustandsvariablen in r, auf die P Bezug nimmt, in r' durch die resultierenden eingeschränkten Zustandsmengen a' und ggf. b' wie in der Aufstellung der Operationen auf unsicherheitsbehafteten Zustandsvariablen ersetzt werden. Für die effiziente operationale Auswertung von Konjunktionen und Disjunktionen ergeben sich folgende Verfahren:

- **Konjunktion**
  Konjunktionen können durch *sequentielle* Auswertung der Argumentprädikate berechnet werden:
  $$(P_1 \wedge P_2)(r) = (P_2 \circ P_1)(r) = P_2(P_1(r))$$
  Diese Regel erlaubt eine sehr effiziente Berechnung von Konjunktionen mit beliebig vielen Termen, ohne besondere Eigenschaften dieser Terme berücksichtigen zu müssen.

- **Disjunktion**

  - Im allgemeinen Fall müssen bei der Berechnung von Disjunktionen die erfüllenden Zustandsteilmengen wieder vereinigt werden:
    $$(P_1 \vee P_2)(r) \supseteq P_1(r) \cup P_2(r)$$
    Die zusätzliche Mengenvereinigung wird hier mittels der |-Operation auf den Datentypen für die einzelnen Zustandsvariablen realisiert. Sie erzeugt bei Prädikaten, die sich auf unterschiedliche Zustandsvariablen beziehen, Ungenauigkeiten aufgrund der bereits diskutierten Kreuzproduktstruktur des Darstellungsraums. Dazu ein Beispiel:
    $$\begin{aligned}
    &(a \stackrel{?}{=} 7 \vee b \stackrel{?}{=} 9)(a = 1...9, b = 8...10) \\
    &= \quad (a = 1..9, b = 8...10) \\
    &\supseteq \quad (a \stackrel{?}{=} 7)(a = 1...9, b = 8...10) \cup (b \stackrel{?}{=} 9)(a = 1...9, b = 8...10) \\
    &= \quad (a = 7, b = 8...10) \cup (a = 1...9, b = 9)
    \end{aligned}$$

  - Wenn von vornherein bekannt ist, daß sich die Terme einer Disjunktion auf *verschiedene* Zustandsvariablen beziehen, kann daher das folgende vereinfachte Berechnungsschema verwendet werden:
    $$(P_1 \vee P_2)(r) = \begin{cases} r & \text{wenn } P_1(r) \neq \emptyset \vee P_2(r) \neq \emptyset \\ \emptyset & \text{sonst} \end{cases}$$

3.3.2 Kernalgorithmus

In diesem Abschnitt wird der Kernalgorithmus der Zustandsexpansion beim *FollowSM*-Verfahren vorgestellt, d. h. die Durchführung der einzelnen Suchschritte im Zustandsraum.

Bisher wurden nur Manipulationen von einzelnen Zustandsbeschreibungen der Form $r = (u_1, ..., u_n)$ behandelt, die aus je einem Unsicherheit modellierenden Zustandsvariablenobjekt pro Zustandsvariable der Protokoll-EFSM bestehen (vgl. Abschnitt 3.3.1). Für solche Zustandsbeschreibungen, abgekürzt mit Z, Z', Z'', wird im folgenden der Einfachheit halber nur noch der Begriff *Zustand* verwendet, obwohl ein solcher Zustand stets für eine *Menge* von Zuständen z der Protokoll-EFSM steht. Ein Zustand Z enthält insgesamt:

- eine repräsentierbare Teilmenge $r \in R$ des Zustandsraums der Protokoll-EFSM,
- einen Zeitgeberzustand τ der OTEFSM, der ja – durch die Intervalle für Zeitgeberabläufe – von Natur aus ebenfalls Unsicherheit beinhaltet,
- das Eintrittsintervall $t_0..t_1$ dieses Zustands, gleichbedeutend mit dem Intervall möglicher Schaltzeitpunkte der vorangegangenen Transition und
- die Indizes i und o der nächsten noch nicht verarbeiteten Eingabe- bzw. Ausgabe-PDU.

In den unten dargestellten Algorithmen wird vereinfachend Z als die repräsentierte Zustandsmenge verwendet.

Die Beweise zu den Theoremen 2 und 5 zeigen, daß die Suche im Zustandsraum auf die elementare Aktion hinausläuft, zu einem gegebenen Zustand Z alle Transitionen zu ermitteln, die aufgrund von Z und den jeweils nächsten noch nicht zugeordneten Kommunikationsprimitiven zulässig sind, und alle sich daraus ergebenden Folgezustände zu berechnen. Die Spezifikation (3.8) der unsicherheitsbehafteten Spuranalyse erlaubt nicht, diese Folgezustände in einem einzelnen – unsicherheitsbehafteten – Zustand zusammenzufassen, weil die i. a. eingeschränkte Repräsentierbarkeit dabei zu Ungenauigkeiten führen könnte, die von der Spezifikation nicht gedeckt sind. Ungenauigkeiten in Form der Obermengenbedingung sind ja nur bei der Ermittlung des Folgezustands im Rahmen der Anwendung *einer* Transition zulässig.

Daher werden in einem Durchlauf der Prozedur *Entwickle*, die einen elementaren Suchschritt modelliert, alle für den unmittelbar nächsten Schaltvorgang in Frage kommenden Transitionen ermittelt und auf die vorliegende Zustandsbeschreibung Z angewandt. Die resultierenden Zustandsbeschreibungen werden in die Liste M eingetragen. Hier folgt der Algorithmus von *Entwickle* als Pseudo-Code. Dabei wird die Abkürzung $P_T(z, i)$ für die *Vorbedingung* für das Schalten einer Transition T verwendet, formal:

$$P_T(z, i) \quad \text{g.d.w.} \quad \exists t(\Phi(T, t, z, i) > (0, 0))$$

Prozedur Entwickle(Z, t_{last})
  Z : Zustand
  t_{last} : spätester Endzeitpunkt des Schrittes

-- unmittelbare Transitionen bearbeiten
$Z'' \leftarrow Z$
Für $T \in \Delta | \psi_T = \perp \wedge \pi_T > 0 \wedge e_T(2) \neq \perp$ mit absteigender Priorität
  Wenn Priorität kleiner als bei letzter T
    $Z \leftarrow Z''$
    Wenn $Z = \emptyset$ **Dann Fertig** -- Fall 1
  $Z' \overset{\supseteq}{\leftarrow} \{z \in Z | P_T(z, \perp)\}$
  Wenn $Z' \neq \emptyset$
    FührеAus$(Z', T, t_0, t_1, t_{\text{last}})$
    $Z'' \overset{\supseteq}{\leftarrow} \{z \in Z'' | \overline{P_T(z, \perp)}\}$ -- andere Transitionen blockiert
$Z \leftarrow Z''$
Wenn $Z = \emptyset$ **Dann Fertig** -- Fall 1

-- verspätete Transitionen bearbeiten
$Z'' \leftarrow Z$
Für $T \in \Delta | \psi_T \neq \perp \wedge \tau_Z(\psi_T)_2 < t_0$ mit absteigender Priorität
  Wenn Priorität kleiner als bei letzter T
    $Z \leftarrow Z''$
    Wenn $Z = \emptyset$ **Dann Fertig** -- Fall 2
  $Z' \overset{\supseteq}{\leftarrow} \{z \in Z | P_T(z, \perp)\}$
  Wenn $Z' \neq \emptyset$
    FührеAus$(Z', T, t_0, t_1, t_{\text{last}})$
    $Z'' \overset{\supseteq}{\leftarrow} \{z \in Z'' | \overline{P_T(z, \perp)}\}$ -- andere Transitionen blockiert
$Z \leftarrow Z''$
Wenn $Z = \emptyset$ **Dann Fertig** -- Fall 2

-- zeitabhängige und spontane Transitionen bearbeiten
$t_{\text{out}} \leftarrow$ Zeit der nächsten Ausgabe-PDU
$t_{\text{m}} \leftarrow \min(t_{\text{out}}, t_{\text{last}})$
Für $t_{\text{e}} : \tau_Z(\psi_{T'})_2$ aktivierter zeitabhängiger $T' \in \Delta$ (aufsteigend) zzgl. t_m
  -- t_{e} sind die möglichen Obergrenzen für den Schaltzeitpunkt
  $Z' \overset{\supseteq}{\leftarrow} \{z \in Z |$ keine nach λ^* höher priorisierte Transition als T' aktiviert bis $t_{\text{e}}\}$
  Wenn $Z' = \emptyset$ **Dann** Weiter
  -- zeitabhängige und spontane Transitionen für Schalten bis t_{e} untersuchen
  Für $T \in \Delta | (\psi_T \neq \perp \wedge \tau_Z(\psi_T)_2 < t_{\text{e}}) \vee \pi_T = 0$
    $Z'' \leftarrow \{z \in Z' | e_T(z, ipdu, opdu)\}$
    Wenn $Z'' \neq \emptyset$
      $[t_0..t_1] \leftarrow$ mögliche Ausführungszeit für T
      -- T ausführen
      Wenn T zeitabhängig **Dann** Zeitgeber zurücksetzen
      $Z'' \overset{\supseteq}{\leftarrow} \alpha_T(Z'', ipdu, opdu)$
      $Z''.t_0 \leftarrow t_0$
      $Z''.t_1 \leftarrow t_1$
      $M \leftarrow M \cup \{Z''\}$
  Wenn t_{e} Endzeit einer Transition
    $Z \overset{\supseteq}{\leftarrow} \{z \in Z | \overline{P_{T'}(z, \perp)}\}$

Wenn $Z = \emptyset$ **Dann Fertig** -- Fall 4

-- Ausgabe- vor nächster Eingabe-PDU ⇒ Sackgasse
Wenn $t_{\mathrm{m}} < t_{\mathrm{last}}$ **Dann Zurück** -- Fall 5

-- Schrittende vor nächster Eingabe-PDU ⇒ diesen Zustand merken
$t_{in} \leftarrow$ Zeit der nächsten Eingabe-PDU
Wenn $t_{\mathrm{last}} < t_{in}$
  $M \leftarrow M \cup \{Z\}$
  Zurück

-- eingabeabhängige Transitionen bearbeiten
$Z'' \leftarrow Z$
Für $T \in \Delta | e_T(2) \neq \{\perp\}$
  Wenn Priorität kleiner als bei letzter T
    $Z \leftarrow Z''$
    Wenn $Z = \emptyset$ **Dann Fertig** -- Fall 3
  $Z' \stackrel{\supseteq}{\leftarrow} \{z \in Z | P_T(z, ipdu)\}$
  Wenn $Z' \neq \emptyset$
    $t_{\mathrm{x}} \leftarrow$ Zeit der Eingabe-PDU
    FührеAus$(Z', T, t_{\mathrm{x}}, t_{\mathrm{x}}, t_{\mathrm{last}})$
    $Z'' \stackrel{\supseteq}{\leftarrow} \{z \in Z'' | \overline{P_T(z, ipdu)}\}$ -- andere Transitionen blockiert
$Z \leftarrow Z''$
Wenn $Z = \emptyset$ **Dann Fertig** -- Fall 3

-- Option: Eingabe-PDU ignorieren, falls von keiner Transition gelesen
Wenn Nicht Option *ignore_ipdu* **Dann Fertig** -- Fälle 6, 7
$Z.t_0, Z.t_1 \leftarrow$ Zeit der nächsten Eingabe-PDU
$M \leftarrow M \cup \{Z\}$

Ende von Entwickle

Die Routine *FührеAus*, die von *Entwickle* verwendet wird, faßt die nötigen Operationen zusammen, um eine Transition T, deren Vorbedingung $P_T(Z, i)$ bereits geprüft wurde, auszuführen, falls ihre Ausgabespezifikation e_T zur nächsten beobachteten Ausgabe-PDU paßt:

Prozedur FührеAus$(Z, T, opdu, ipdu)$
  Z : Zustand
  T : Transition (Vorbedingung $P_T(Z, ipdu)$ erfüllt)
  $opdu$: nächste Ausgabe-PDU
  $ipdu$: nächste Eingabe-PDU

  $Z \leftarrow \{z \in Z | e_T(z, ipdu, opdu)\}$
  Wenn $Z \neq \emptyset$
    $[t_0..t_1] \leftarrow$ mögliche Ausführungszeit für T
    -- T ausführen
    Wenn T zeitabhängig **Dann** Zeitgeber zurücksetzen

$Z \overset{\supseteq}{\Leftarrow} \alpha_T(Z, ipdu, opdu)$
$Z.t_0 \leftarrow t_0$
$Z.t_1 \leftarrow t_1$
$M \leftarrow M \cup \{Z\}$

Ende von FührеAus

Der Algorithmus berechnet alle möglichen Entwicklungsschritte für jeweils maximale Schaltzeitintervalle $[t_0..t_1]$. Die relativ komplizierten Priorisierungsregelungen erfordern, daß die aktuelle Zustandsbeschreibung Z nach jeder Klasse gleichpriorisierter Transitionen um diejenigen elementaren Zustände z gekürzt wird, in denen ein Schaltvorgang erzwungen und die restlichen Transitionen dadurch blockiert werden. Besonders kompliziert gestaltet sich die Prioritätsauswertung für zeitabhängige Transitionen, weil die Priorisierung nur unter zeitabhängigen Transitionen *mit demselben auslösenden Zeitgeber* gilt und weil mit jedem Zeitgeberablauf eine neue Blockierungssituation eintritt, aber nur für diejenigen Transitionen T und Elementarzustände z, deren Aktivierungsprädikat e_T durch z erfüllt wird. Daher müssen für unterschiedliche Zustandsteilmengen u. U. unterschiedliche Schaltzeitintervalle für zeitabhängige und spontane Transitionen verarbeitet werden.

Der Zeitpunkt t_{last} begrenzt das betrachtete Zeitintervall nach oben, damit im in Abschnitt 2.2.1 geforderten „Online"-Betrieb keine Blockierung durch ausbleibende PDUs eintreten kann: Wenn zur Zeit t_{last} keine weiteren PDUs eingetroffen sind, wird der gerade entwickelte Zustand unverändert als „neue" Zustandsbeschreibung übernommen. Dies ist formal unproblematisch, sofern t_{last} weiter in der Zukunft liegt als die maximale Ausgabeverzögerung d_T sämtlicher Transitionen.

3.3.3 Suchstrategie

Im letzten Abschnitt wurde die Berechnung eines einzelnen Suchschritts beschrieben. Dabei hat noch keine Festlegung stattgefunden auf die zu verwendende Suchstrategie bei der Konstruktion des Zustandssuchbaums. Klassischerweise kommen hier einerseits die *Tiefensuche* (*depth-first search*, DFS) und andererseits die *Breitensuche* (*breadth-first search*, BFS) in Frage. Diese Entwurfsentscheidung schlägt sich vordergründig in der Auswahl des jeweils nächsten zu entwickelnden Zustands Z aus der Zustandsliste M nieder.

In den Arbeiten [EvB95] und [BvBDS91] wird die Tiefensuche favorisiert, weil bei der Breitensuche alle noch nicht entwickelten Knoten des Suchbaums dieselbe Tiefe im Baum haben und daher im Speicher aufbewahrt werden müssen. Bei der Tiefensuche genügt es dagegen, den aktuell bearbeiteten Suchpfad zu speichern ab dem ältesten Knoten, von dem aus noch eine Verzweigung möglich ist (Rücksetzpunkt). Wenn dieser älteste Rücksetzpunkt weit genug zurückliegt, wird er üblicherweise gelöscht, weil ein vollständiges Scheitern des aktuell untersuchten Zweiges heuristisch oder formal aufgrund der Protokolleigenschaften ausgeschlossen wird. Damit wird eine Aufwandsbegrenzung erreicht durch Abschneiden – *hoffentlich* – redundanter Entwicklungspfade.

Im *FollowSM*-System wird dagegen der Ansatz der Breitensuche verfolgt. Für diese Entwurfsentscheidung sprechen – auch unter Berücksichtigung der genannten Argumente für die Tiefensuche – folgende Punkte:

1. Bei der Breitensuche entspricht die Entwicklungsreihenfolge im wesentlichen der Abfolge der beobachteten Ein- und Ausgabenachrichten. Daraus folgt eine hohe Lokalität der Zugriffe auf die Nachrichtenfolge und die Möglichkeit, eintreffende Nachrichten sofort und endgültig zu verarbeiten. Es ist nicht wie bei der Tiefensuche erforderlich, Nachrichten zu speichern, um beim Zurückrollen der Suche bis zu einem älteren Rücksetzpunkt die zugehörigen Nachrichten wieder zur Verfügung zu haben. Damit eignet sich die Breitensuche besonders gut für den „Online"-Betrieb.

2. Nur die Breitensuche erlaubt die *Pfadvereinigung* ohne Mehraufwand. *FollowSM* macht regen Gebrauch von der Möglichkeit, daß ein Protokollzustand z über verschiedene Teilpfade der Zustandsentwicklung erreicht werden kann. Wann immer eine resultierende Zustandsbeschreibung Z

   (a) identisch ($Z = Z'$) zu,

   (b) eine Verallgemeinerung ($Z \supset Z'$) oder

   (c) eine Spezialisierung ($Z \subset Z'$)

   einer schon erhaltenen Zustandsbeschreibung Z' ist, so braucht einer von beiden Pfaden nicht weiterentwickelt zu werden, weil seine Elementarzustände sonst doppelt untersucht würden. Im den Fällen 2a und 2c entfällt Z, im Fall 2b wird Z' durch Z ersetzt. Diese Vorgehensweise ist in den Prozeduren *Entwickle* und *FührеAus* in Abschnitt 3.3.2 durch Terme der Form $M \cup \{Z\}$ wiedergegeben.

   Die Pfadvereinigung reduziert den Rechen- und Speicheraufwand um Größenordnungen, weil jedes der genannten Ereignisse einen ganzen Unterbaum im Suchbaum abschneidet. Figur A.15 zeigt ein Beispiel, in dem sich der abgeschattet dargestellte Teilbaum durch eine einzige Pfadvereinigung als redundant herausstellt. Bei der Tiefensuche wäre die Pfadvereinigung nur möglich, wenn der *gesamte* Suchbaum ab dem ältesten Rücksetztpunkt ständig gespeichert bleibt – ein vom Speicheraufwand völlig unpraktikables Vorgehen.

3. Man kann auf Heuristiken verzichten, die angeben, wie lange ein Rücksetzpunkt aufgehoben werden muß. Damit ist die Breitensuche auch das formal vollständigere bzw. protokollunabhängige Verfahren.

Natürlich sind der Rechen- *und* Speicheraufwand für einen Suchbaum im schlimmsten Fall grundsätzlich exponentiell in der Suchtiefe, hier also in der Länge der beobachteten Nachrichtensequenz. Für *FollowSM* gilt jedoch, daß das jeweils vorliegende zu analysierende Protokoll dank der Pfadvereinigung eine faktische obere Schranke setzt für die Anzahl der gleichzeitig zu speichernden alternativen Zustandsbeschreibungen. Mit dieser oberen Schranke, etwa m, ergeben sich dann:

- ein in der Beobachtungslänge *linearer* Aufwand für die Rechenzeit, da pro Entwicklungsschritt höchstens m-mal *Entwickle* ausgeführt werden muß und m höchstens linear in die Ausführungszeit von *Entwickle* eingeht (insbesondere beim Test auf redundante Zustände).

- ein *konstanter* Speicheraufwand von m Zustandsbeschreibungen.

Daraus ergibt sich als Fazit, daß die *Breitensuche mit Pfadvereinigung* der Tiefensuche in allen qualitativen und quantitativen Belangen überlegen ist.

Die Realisierung der Breitensuche in der Prozedur *Analysiere* ist im folgenden skizziert. Die als Parameter übergebene Zeit t_{to} dient wieder zur Begrenzung des Entwicklungsintervalls vor allem für die „Online"-Analyse. Der Algorithmus zeigt auch das Kriterium für das Erkennen eines Verhaltensfehlers: Wenn die Zustandsliste M nach einem Entwicklungsschritt leer ist, gibt es keinen Zustand mehr, der mit der Beobachtung nach Maßgabe des Protokolls vereinbar ist (vgl. Theoreme 3, 4 und 5). Es muß sich also – spätestens zum aktuellen Zeitpunkt der Entwicklung – ein fehlerhaftes Verhalten der beobachteten Instanz ereignet haben.

```
Variablen:
  M    : Zustandsliste

  Prozedur Analysiere(t_to)
    t_to          : Soll-Ende des Analyse-Abschnitts

    Wiederhole
      Wenn M = ∅
        M ← Initialzustand zur Zeit Z.t_1
      Z ← Z' ∈ M|Z'.t_0 minimal
      Abbruch Wenn Z.t_0 > t_to
      M ← M − Z
      Entwickle(Z, Z.t_1 + d_max)
      Wenn M = ∅
        Ausgabe: Verhaltensfehler (Fall 1..7)

  Ende von Analysiere
```

Bei der Initialisierung der Zustandsliste M müssen im allgemeinen mehrere Zustandsobjekte Z eingetragen werden, weil zu jedem Z ja die Verweise auf die nächsten PDUs gehören. Im OTEFSM-Konzept können Ausgaben ja mit gewisser Verspätung gegenüber den Schaltereignissen beobachtet werden. Daher können alle Ausgaben, die zwischen dem Beobachtungsbeginn oder -neubeginn und dem Maximum d_{max} der Ausgabeverzögerungen der Transitionenbeobachtet werden,

$$d_{max} = \max_{T \in \Delta} d_T,$$

von einem Schaltereignis *vor* dem Beobachtungsbeginn stammen. Ausgaben, für die das gilt, müssen ignoriert werden, damit nicht fälschlicherweise Verhaltensfehler gemeldet werden. Figur A.16 zeigt an einem Beispiel, wie sich dies auf die initiale Zustandsliste M auswirkt.

3.3.4 Module und Unabhängigkeit

Bisher war ausschließlich von der Spuranalyse einer einzelnen Protokoll-EFSM die Rede, modelliert durch eine einzelne OTEFSM. Meistens sind praktische Protokolle zwar im wesentlichen mit einer einzelnen EFSM spezifiziert, in einer realen Implementierung arbeiten aber fast immer mehrere Protokoll-EFSMs nebeneinander. Denn

1. Protokolle ab OSI-Schicht 3 (Netzwerkschicht) aufwärts können meist mehrere virtuelle Verbindungen (*connections*) oder mehrere Sitzungen (*sessions*) unterhalten, wobei jede Verbindung oder Sitzung durch eine eigene EFSM beschrieben wird, und

2. die Protokolle mehrerer übereinanderliegender Schichten werden ebenfalls durch getrennte EFSMs modelliert.

Um den *FollowSM*-Analysator in diesen Fällen sinnvoll einsetzen zu können, muß er mehrere unsicherheitsbehaftete OTEFSMs gleichzeitig analysieren können. Dies stellt keinerlei zusätzliches Problem dar, weil die Zustände aller beteiligten OTEFSMs, wie in Abschnitt 3.3.3 dargestellt, in einer gemeinsamen Zustandsliste M verwaltet werden können, deren ältester Eintrag im jeweils nächsten Suchschritt weiterentwickelt wird. Dasselbe Verfahren erlaubt auch, beide Endpunkte einer Kommunikationsverbindung gleichzeitig zu überwachen, wodurch die in Abschnitt 2.2.1 begründete Entscheidung für *eine* beobachtete Instanz folgenlos bleibt. Die einzelnen OTEFSM-Instanzen mit separaten Zustandsbeschreibungen heißen bei *FollowSM*, in Anlehnung an Estelle, *Module*.

Allerdings ist die gleichzeitige Analyse mehrerer OTEFSMs nicht gleichwertig zu einem vollständigen CEFSM-Modell (kommunizierende EFSMs, *communicating* EFSMs) wie z. B. in SDL [CCI87]: Dort kann *jede* EFSM mit jeder anderen der Spezifikation kommunizieren, während in *FollowSM* alle OTEFSM-Instanzen unabhängig voneinander ausschließlich über die beobachtete Kommunikationsstrecke Nachrichten austauschen. Figur A.17 skizziert den Unterschied.

Wesentliches Merkmal der Verwendung mehrerer Module in *FollowSM* ist deren zwangsläufige *Unabhängigkeit*: Keine in einem Modul getroffene Zustandsentscheidung kann irgendeine Auswirkung haben auf Zustandsentscheidungen in anderen Modulen. Formal ausgedrückt ist die Menge der aufgrund der Unsicherheit möglichen Systemzustände zu jedem Zeitpunkt das Kreuzprodukt der möglichen Zustände jedes einzelnen Moduls.

Beispiel: Im B-ISDN-Signalisierungsprotokoll Q.2931 [IT94b] existieren genau ein Systemprozeß je Implementierung und ein verbindungsspezifischer Prozeß je *call reference*. Der Systemprozeß kann eine Operation ausführen, in der alle bestehenden Verbindungen abgebaut, d. h. alle verbindungsspezifischen Prozesse terminiert werden. Wenn der Erfolg einer solchen Operation vom Systemzustand abhängt, gibt es folgende Möglichkeiten:

1. Der Abbau findet statt $\Rightarrow$ alle Verbindungsprozesse terminieren.

2. Der Abbau findet nicht statt $\Rightarrow$ kein Verbindungsprozeß terminiert.

Bei *FollowSM* kann diese Alles-oder-nichts-Entscheidung nicht realisiert werden, es müßte jedes verbindungsspezifische Modul selbst über den eigenen Abbau entscheiden, was bei unsicherer Zustandsinformation zu jeder denkbaren Kombination aus abgebauten und nicht nicht abgebauten Einzelverbindungen führen könnte.

Diese *Unabhängigkeitsprämisse* führt also offenbar zu Ungenauigkeiten in der Modellierung. Allerdings bleibt die – dort nur pro Modul erhobene – Forderung aus Abschnitt 3.2.2 nach Korrektheit erfüllt, weil bei Vernachlässigung möglicher Abhängigkeiten eventuell zu viele, aber nie zu wenige Systemzustände betrachtet werden. Außerdem wäre ein Verzicht auf die

Unabhängigkeitsprämisse mit einer nicht hinnehmbaren Erhöhung des Zeit- und Speicheraufwands verbunden: *Jede* Kombination zwischen möglichen Modulzuständen müßte als einzeln zu verarbeitender Systemzustand verwaltet werden. Tabelle 3.2 zeigt einen Vergleich der Variantenzahlen, die sich mit und ohne Unabhängigkeitsprämisse ergeben.

3.3.5 Systemstruktur

Dieser Abschnitt erläutert grob die Datenflüsse und die Systemstruktur, die dem *FollowSM*-Analysator zugrundeliegen. Figur A.18 faßt diese Informationen grafisch zusammen.

Leser

Die Schnittstelle zum beobachteten Kommunikationsmedium bildet eine *Leser*-Komponente, die die ausgetauschten PDUs mit Zeitstempeln als Bitfolgen bereitstellt. Allein der Austausch dieses Lesers, der im allgemeinen nur aus wenigen Code-Zeilen besteht, entscheidet über den „Online"- oder „Offline"-Betrieb oder erlaubt den Wechsel der physikalischen Schnittstelle oder – beim „Offline"-Betrieb – des Dateiformats.

Dekoder

Die nächste Hauptkomponente ist der *Dekoder*, der die vom Leser in Gestalt einer Bitkette bereitgestellten PDUs in eine Folge von Attributwerten überführt, die den einzelnen PDU-Feldern entsprechen. Diese Darstellung sollte soweit wie möglich an der *Semantik* der PDU-Inhalte im Rahmen des Zielprotokolls orientiert sein, d. h. rein syntaktische Freiheiten in der PDU-Kodierung, die für das Protokollverhalten bedeutungslos sind, gar nicht erst in die interne Attributdarstellung eingehen lassen. Dies betrifft z. B. Freiheiten in der Reihenfolge von Datenfeldern oder Informationen, die nur die Grenzen eines folgenden Feldes innerhalb der Bitkette definieren. Nicht vorhandene Optionale PDU-Felder können vom Dekoder ggf. schon durch die im Protokoll für diesen Fall festgelegten Standardwerte ersetzt werden.

Weil bekannte Spezifikationssprachen für die PDU-Syntax wie z. B. ASN.1 dieser Aufgabe der semantikerhaltenden Reduktion für beliebige konkrete Transfersyntaxen (vgl. Abschnitt 2.1.2) nicht gewachsen sind, wird die PDU-Syntax im *FollowSM*-Ansatz nicht mit Hilfe einer Spezifikationssprache beschrieben, sondern der Dekoder prozedural entworfen.

Der Dekoder muß außerdem erkennen, zu welcher Art von OTEFSM-Modul und zu welcher OTEFSM-Instanz dieser Art eine beobachtete PDU gehört, damit sie für die Verhaltensanalyse richtig zugeordnet wird. Fehlerhaft kodierte PDUs werden ebenfalls vom Dekoder erkannt.

Analysator und OTEFSM-Instanzen

In der *Analysator*-Komponente ist die Logik der unsicherheitsbehafteten Spuranalyse zusammengefaßt. Diese Komponente ist vollkommen protokollunabhängig. Instanzen der protokollspezifischen OTEFSM-Module werden immer dann erzeugt, wenn PDUs gelesen werden, die der Dekoder einem noch nicht existierenden Modul zuordnet. Sie werden entfernt, sobald sie in einem sicheren inaktiven Zustand sind und keine ihnen zugeordneten PDUs mehr vorliegen.

Name, Parameter	**Wertebereich**	**Darstellung**	**Semantik**
BOOL	$\{\mathrm{W}, \mathrm{F}\}$	$a \in \{\mathrm{W}, \mathrm{F}, \omega\}$	$\{a\}$ wenn $a \neq \omega$ $\{\mathrm{W}, \mathrm{F}\}$ wenn $a = \omega$
INT	$\mathbb{Z}$	$a \in \mathbb{Z} \cup \{\omega\}$	$\{a\}$ wenn $a \neq \omega$ $\mathbb{Z}$ wenn $a = \omega$
$\mathrm{RANGE}_{L,U}$ $L, U \in \mathbb{Z}$ $L < U$	$\{L, ..., U\}$	$a, b \in \{L, ..., U\}$	$\{a, ..., b\}$ wenn $a \leq b$ $\{L, ..., b, a, ..., U\}$ wenn $a > b$
$\mathrm{SET}_{L,U}$ $L, U \in \mathbb{Z}$ $L < U$	$\{L, ..., U\}$	$a \in \{0,1\}^{U-L+1}$	$\{x \in \{L, ..., U\} \mid a_{x-L} = 1\}$
$\mathrm{LSET}_{L,U}$ $L, U \in \mathbb{Z}$ $L < U < L + 32$	$\{L, ..., U\}$	$a \in \{0,1\}^{U-L+1}$	$\{x \in \{L, ..., U\} \mid a_{x-L} = 1\}$

Tabelle 3.1: Die *FollowSM*-Datentypen zur unsicherheitsbehafteten Modellierung von Zustandsvariablen.

Anzahl ...	**Allgemein**	**Beispiel**
Module	m	10
Zustände pro Modul	n	3
Modulzustände (unabhängig)	$m \cdot n$	$3 \cdot 10 = 30$
Systemzustände (abhängig)	n^m	$3^{10} = 59049$

Tabelle 3.2: Zustandsexplosion bei Verzicht auf die Unabhängigkeitsprämisse.

3.3.6 Analyseberichte

Dieser Abschnitt erläutert, welche Ergebnisse der *FollowSM*-Prototyp dem Anwender liefert. Zu den unterschiedlichen Arten von abrufbaren Analyseberichten gehören jeweils spezifische Verfahren, wie die im Suchbaum der Spuranalyse enthaltenen Informationen ausgewertet und aufbereitet werden. Figur A.19 stellt die vollständige Hierarchie von *FollowSM*-Analyseberichten dar, in der die Pfeilrichtung für Abstraktion und damit Verdichtung eines Ursprungsberichtes steht. Die Zahlen an den rechten unteren Ecken der Kästchen geben die Länge der Berichte in Zeilen für ein konkretes Beispiel an, um die Größenordnung dieser Informationsverdichtung zu illustrieren. Die einzelnen Berichtstypen werden im folgenden erklärt.

Protokoll aller Suchschritte

Hier werden alle einzelnen Suchschritte der Zustandsentwicklung – einschließlich derjenigen Pfade, die später als Sackgassen enden – ausgegeben. Es handelt sich also um eine vollständige Darstellung des Suchbaums. Diese ist aufgrund zahlreicher sich später widerlegender Verzweigungen sehr unübersichtlich und eignet sich kaum für einen Menschen als Leser.

Relinearisierte Abfolge der Zustände

Hier wird eine Folge von Zuständen und Transitionen rekonstruiert, die das beobachtete Verhalten der IUT erklärt, also ein einzelner nichtabbrechender Pfad im Suchbaum. Diese Folge ermöglicht dem Benutzer einen guten Einblick in die abgelaufene Kommunikation und kann, anders als der vollständige Suchbaum, schon eine wertvolle Hilfestellung sein.

Bei der Ermittlung der Zustandsfolge sind zwei miteinander verwandte Teilprobleme zu lösen:

1. Durch die Pfadvereinigungen, wie in Abschnitt 3.3.3 beschrieben, verwandelt sich der Suchbaum in einen gerichteten Graphen, der die Baumeigenschaft eindeutiger Vorgängerknoten nicht mehr aufweist. Es wird eine Datenstruktur benötigt, die diesen Suchgraphen verwalten und daraus effizient den „richtigen" erklärenden Pfad ableiten kann.

2. Jede durch eine Pfadvereinigung entstandene Masche im Suchgraphen führt zu einer Mehrdeutigkeit in der Zustandsentwicklung. Da im Interesse der Benutzerfreundlichkeit ein *einzelner* erklärender Pfad ermittelt werden soll, müssen diese Mehrdeutigkeiten aufgelöst werden. Außerdem soll zu jedem Zustand der Zustandsfolge eine Angabe erfolgen, ob er mit Sicherheit oder – aufgrund einer solchen Mehrdeutigkeit – nur vielleicht von der IUT durchlaufen wurde.

Die Auflösung von Mehrdeutigkeiten ist in den Fällen vorgegeben, wo eine Verallgemeinerung oder Spezialisierung einer bekannten Zustandsbeschreibung stattfindet: Die allgemeinere Zustandsbeschreibung kann *alle* korrekten zukünftigen Verhaltenalternativen erklären, weshalb der zu ihr führende Pfad statt des anderen in die erklärende Zustandssequenz aufgenommen werden muß. Bei gleichen Zustandsbeschreibungen findet eine nichtdeterministische

Auswahl statt, die in *FollowSM* durch Beibehalt des zuerst gefundenen Pfades als einfachere Variante realisiert ist.

Die *Pfad*-Datenstruktur des *FollowSM*-Systems zur Verwaltung des Suchgraphen basiert auf einem gerichteten Graphen mit zwei Kanten- und drei Knotentypen. Die Kantentypen sind:

1. **Primäre Kanten**: Eine primäre Kante bildet jeweils die allgemeinste Erklärung für das Zustandekommen einer Zustandsbeschreibung. Jeder Zustandsknoten hat höchstens eine hinführende Primärkante. Folglich spannen die Primärkanten einen Baum auf, der das Zustandekommen seiner Blattknoten vollständig erklärt, den *Primärbaum*

2. **Sekundäre Kanten**: Sekundäre Kanten markieren alternative Zustandsübergänge, die zumindest eine Teilmenge der Zustandsbeschreibung ihres Zielknotens erzeugt haben könnten. Knoten können mehrere hinführende Sekundärknoten aufweisen.

An Knotentypen gibt es:

1. **Blattknoten**: Blattknoten sind die Blätter des Primärbaums. Sie tragen die Zustandsbeschreibungen, die noch nicht weiterentwickelt worden sind.

2. **Entwickelte Knoten**: Knoten des Entwicklungsgraphen, die mindestens einen über eine Primär- oder Sekundärkante erreichbaren Nachfolger aufweisen. Folgende Unterarten entwickelter Knoten existieren:

   (a) **Als „unsicher" markierte** Knoten tragen Zustandsbeschreibungen, die nur vielleicht von der IUT durchlaufen wurden.

   (b) **Unmarkierte** Knoten sind noch nicht als unsicher markiert. Dies ist der „Urzustand" eines entwickelten Knotens, Änderungen kommen nur in der Richtung *unmarkiert* zu *unsicher* vor. Weiter unten wird ausgeführt, wann der Schluß gültig ist, daß die zugehörige Zustandsbeschreibung sicher durchlaufen wurde.

Außerdem gibt es stets zwei ausgezeichnete Knotenpositionen im Primärbaum, die als u-Position und v-Position bezeichnet werden. Für die Definition dieser Positionen wird im folgenden von „früheren" und „späteren" Knoten gesprochen, wobei die Kanten von früheren zu späteren verlaufen.

- u markiert den frühesten Knoten, von dem mehr als eine Kante ausgeht.
- v markiert den frühesten Knoten, von dem mehr als eine Primärkante ausgeht.

Daraus folgt $u \leq v$ im Sinne des „Früher"-Begriffs. Bis zur Position u liegt bereits eine reine Zustandssequenz vor, zwischen u und v bilden nur noch die Primärkanten eine verzweigungsfreie Sequenz. Außerdem gilt die Invariante, daß alle Knoten bis einschließlich v höchstens einen Vorgänger haben, mit anderen Worten keine hinführenden Sekundärkanten aufweisen.

Figur A.20 illustriert die *Pfad*-Datenstruktur des *FollowSM*-Systems zur Verwaltung des Suchgraphen anhand eines Beispiels mit zwei Operationen. Alle Kanten verlaufen von links nach rechts. Graph a zeigt die Ausgangssituation. Als erste Operation wird eine Pfadvereinigung aufgrund eines von Knoten 1 erreichbaren Spezialfalls von Knoten 2 durchgeführt

(b). Der primäre Pfad zu 1 wird daher von rechts nach links in sekundäre Kanten transformiert, bis hinunter zum v-Knoten, weil dieser erstmals weitere Primärnachfolger aufweist. Die zweite Operation ist das Löschen des Knotens 3, nachdem dessen Entwicklung ohne Folgezustände in einer Sackgasse endet (c). Alle inneren Knoten ohne Nachfolger müssen entfernt werden, wobei der in c gezeigte Graph übrigbleibt. Weil auch die letzte alternative ausgehende Primärkante vom v-Knoten gelöscht wurde, rückt v zwei Positionen nach rechts. Nun muß aber noch die v-Invariante wiederhergestellt werden, indem die Sekundärpfade zum neuen v- und dessen Vorgängerknoten gelöscht werden, wobei ein weiterer Knoten wegfällt (d). Dabei werden alle Knoten in den Maschen aus dem Primär- und den gelöschten Sekundärpfaden als „unsicher" markiert, weil die gelöschten Sekundärpfade ja mögliche Alternativerklärungen darstellen. Anschließend rückt die u-Position noch bis zur neuen ersten Sekundärabzweigung vor.

Alle Knoten ab u nach links stellen die Zustandssequenz dar, die ausgegeben werden soll. Nicht als „unsicher" markierte Knoten sind hier mit Sicherheit von der IUT durchlaufen. Zwischen u und v stehen zwar schon die Knoten der gesuchten Sequenz fest, aber noch nicht das Unsicherheitsattribut. Dies erkennt man im Teilbild d von Figur A.20: Das endgültige Unsicherheitsattribut am Nachfolger des u-Knotens hängt davon ab, ob der untere Blattknoten erfolgreich weiterentwickelt werden kann (geschlossene Sekundärmasche) oder nicht (Sekundärpfad wird gelöscht).

Figur A.21 stellt die Struktur des Analyseberichts *Zustandsfolge* für den Fall einer Synchronisationsphase dar. Man erkennt die beobachteten PDU-Typen, die rekonstruierte Transitionsfolge und die Entwicklung dreier unsicherheitsbehafteter Zustandsvariablen, nämlich des Basiszustands, eines Zeitgebers und eines Sequenzzählers. Nach nur sieben PDUs sind die Werte dieser drei Variablen sicher ermittelt.

Fehlerereignisse mit Vorgeschichte

Wenn die beobachtete Kommunikation arm an Fehlersituationen ist, dann wirkt auch die Länge der relinearisierten Zustandsentwicklung störend. Daher kennt *FollowSM* einen Analysebericht, der sich auf die Fehlerereignisse der beobachteten Kommunikation beschränkt, wobei jedes Fehlerereignis mit dem unmittelbar vorausgegangenen Teilstück der Zustandsfolge gemäß dem vorigen Abschnitt in einer konfigurierbaren Anzahl von Entwicklungsschritten beschrieben wird.

Die Fehlerereignisse selbst werden mit einer protokollunabhängig automatisch generierten Meldung beschrieben, die davon abhängt, an welcher Stelle der Prozedur *Entwickle* die Zustandsentwicklung des letzten vorhandenen Blattzustands erfolglos abgebrochen wurde. Diese Stellen sind im Algorithmus in Abschnitt 3.3.2 als Fall 1 bis Fall 7 gekennzeichnet und führen sinngemäß zu folgenden Meldungen:

Fall 1 Die aktivierte unmittelbare Transition ... hat gemäß der beobachteten Ausgaben nicht geschaltet.

Fall 2 Die aktivierte zeitabhängige und aufgrund des Zeitgeberablaufs verspätete Transition ... hat gemäß der beobachteten Ausgaben nicht geschaltet.

Fall 3 Die eingabeabhängige Transition ... hat trotz passender Eingabe nicht geschaltet.

Fall 4 Die aktivierte zeitabhängige Transition ... blockiert alle alternativen Transitionen, weil Zeitgeber ... abgelaufen ist.

Fall 5 Es ist keine Transition aktiviert, die die zur Zeit ... beobachtete Ausgabe-PDU ... erzeugen könnte.

Fall 6 Es ist keine Transition aktiviert, die die zur Zeit ... beobachtete Eingabe-PDU ... lesen könnte.

Fall 7 Keine der aktivierten Transitionen, die die zur Zeit ... beobachtete Eingabe-PDU ... lesen konnten, hat gemäß der beobachteten Ausgaben geschaltet.

Natürlich sagt die jeweilige Meldung nur etwas über das Scheitern der letzten Zustandsexpansion aus. Eine tatsächliche Fehlfunktion der IUT, die zu einer Inkonsistenz zwischen ihrem internen Zustand und ihrem Verhalten geführt hat, kann sich jedoch schon früher ereignet haben, sowohl bei einem Vorgänger der Zustandsbeschreibung, auf der die Meldung basiert, als auch innerhalb eines Alternativzweigs. Zuverlässigere Angaben zur Fehlerursache sind anhand der Protokollspezifikation nicht ableitbar, weil ja keinerlei Informationen zur Implementierung der IUT und zu ihren internen Zuständen während der Spurerzeugung zugrundegelegt werden.

Hinzu kommen noch Kodierungsfehler in Ausgabe-PDUs, die vom Dekoder gemeldet werden.

Fehlerklassifikation und Schätzung des Lasteinflusses

In Abschnitt 2.2.1 wurde erläutert, warum aus Anwendersicht eine automatische Zusammenfassung der *Fehlerereignisse* gemäß vorigem Abschnitt zu *Fehlerklassen* sehr wünschenswert ist. Diese Zusammenfassung – oder *Fehlerklassifikation* – benötigt eine Ähnlichkeitsregel, die angibt, wann genau zwei Fehlerereignisse derselben Fehlerklasse zugerechnet werden sollen, weil sie wahrscheinlich eine gemeinsame *Ursache* haben – z. B. einen Implementierungsfehler in der IUT oder eine bestimmte Überlastsituation.

Für *FollowSM* gehören zwei Fehlerereignisse derselben Klasse an, wenn sie übereinstimmen,

- bei Kodierungsfehlern, die der Dekoder feststellt, im Typ der betroffenen PDU und der Fehlermeldung des Dekoders, bzw.
- bei Verhaltensfehlern, im Abbruchfall (Fälle 1 bis 7 oben) und der Sequenz der letzten n erfolgreichen Transitionen in der relinearisierten Zustandsentwicklung.

Die Schwelle n ist vom Anwender konfigurierbar.

Weiteren Aufschluß über mögliche Fehlerursachen erlaubt die Unterscheidung zwischen permanenten, transienten und intermittierenden Fehlern. Zu diesem Zweck müssen neben den Fehlerereignissen jeder Fehlerklasse auch die zugehörigen *Gutfälle* identifiziert werden. Das sind Ereignisse, in denen die der Fehlerklasse entsprechende Kommunikationssituation vorliegt, ohne daß das jeweilige Fehlverhalten aufgetreten ist. Die Kommunikationssituation der Fehlerklasse definiert sich

- im Falle von Kodierungsfehlern durch den PDU-Typ,
- im Falle von Verhaltensfehlern durch die charakteristische n-elementige Transitionssequenz

der Fehlerklasse. Mit Hilfe der Anzahl e_c der Fehlerfälle, der Anzahl g_c der Gutfälle und der Anzahl s_c der Eintritte der Kommunikationssituation der Fehlerklasse c in einem Zeitraum ergibt sich ihre *Fehlerquote* q_c zu

$$q_c \stackrel{def}{=} \frac{e_c}{s_c} = \frac{e_c}{g_c + e_c}. \tag{3.9}$$

Implementierungsfehler werden vorwiegend permanente Fehler hervorrufen, bei denen die Fehlerquote nahe bei 1 liegt. Überlastsituationen und Störsignale auf dem Medium dagegen werden vorwiegend zu transienten und intermittierenden Fehlern führen, für deren Fehlerklassen sowohl Fehler- als auch Gutfälle auftreten.

Lastabhängige Fehler, die durch überhöhte Kommunikationslast begünstigt oder verursacht werden, sind besonders häufige Problemquellen in realen Kommunikationssystemen. Daher kann *FollowSM* für jede Fehlerklasse die *Lastkorrelation* der Fehlerquote berechnen. Dazu wird die Beobachtungszeit in Intervalle eingeteilt und für jedes Intervall i und jede Fehlerklasse c die Fehlerquote q_c^i sowie die *Last* l^i berechnet. l^i ist die Anzahl der PDUs, die derjenigen OSI-Protokollschicht zugeordnet wurden, aus der die Fehlerklasse c stammt, und deren Beobachtungszeitstempel im Intervall i liegen. Die Länge der Zeitintervalle für die Korrelationsbestimmung kann vom Anwender konfiguriert werden. Die Lastkorrelation ld_c ist der Korrelationskoeffizient der Paare (l_i, q_c^i) von Last und Fehlerquote, bei m Intervallen:

$$ld_c \stackrel{def}{=} \frac{\sum_{i=1}^{m} \left(l^i - \frac{1}{m} \sum_{j=1}^{m} l^j \right) \left(q_c^i - \frac{1}{m} \sum_{j=1}^{m} q_c^j \right)}{\sqrt{\sum_{i=1}^{m} \left(l^i - \frac{1}{m} \sum_{j=1}^{m} l^j \right)^2 \sum_{i=1}^{m} \left(q_c^i - \frac{1}{m} \sum_{j=1}^{m} q_c^j \right)^2}} \tag{3.10}$$

Im Analysebericht der Fehlerklassen werden keinerlei ereignisbezogene Informationen mehr ausgegeben. Stattdessen enthält der Bericht pro aufgetretener Fehlerklasse die im folgenden noch einmal zusammengefaßten Angaben:

- Beschreibung der Fehlerklasse durch Fehlerart und ggf. charakteristische Transitionssequenz.
- Anzahl der Fehlerfälle während der Beobachtung.
- Anzahl der Gutfälle während der Beobachtung seit dem ersten Fehlerfall.
- Fehlerauftrittsquote als Verhältnis aus Fehler- zu Fehler- und Gutfällen seit dem ersten Fehlerfall.

- Lastkorrelation der Fehlerquote.
- Anzahl der Zeitintervalle, die in die Berechnung der Lastkorrelation eingegangen sind.

Alles in allem liefert die statistische Fehlerauswertung ein kompaktes, leichter interpretierbares Ergebnisprotokoll, das nach Ursachen, nicht nach Symptomen strukturiert ist.

Protokollparameter-Schätzung

Die in Abschnitt 2.2.1 beschriebene Schätzung von Protokollparametern erfolgt mit Hilfe einer pro OTEFSM-Klasse vorhandenen protokollspezifischen Prozedur, die den Transitionstyp und den Ausgangs- und Endzustand jedes Zustandsübergangs verarbeitet, der in der *relinearisierten* Zustandsentwicklung *sicher* stattgefunden hat. Welche Rückschlüsse auf Protokollparameter möglich sind, hängt vom Protokoll ab und läßt sich nicht aus einer Protokollspezifikation ableiten, die *alle* denkbaren Parametrisierungen zulassen muß, um keine unzutreffenden Meldungen über Protokollverstöße zu produzieren.

Eine prinzipielle Alternative zur Ermittlung von Protokollparametern besteht darin, eine Zustandsvariable mit dem Wertebereich der möglichen Werte eines Parameters einzuführen und die Verhaltensentscheidungen im OTEFSM-Modell vom Wert dieser Zustandsvariable abhängig zu machen, der nie explizit geändert wird. Im Rahmen der unsicherheitsbehafteten Spuranalyse wird diese Parametervariable als „ungewiß" initialisiert. Wenn sie später auf einen sicheren Wert eingeschränkt wird, ist nur dieser Wert des Parameters mit dem beobachteten Verhalten verträglich. Dieses Vorgehen ist jederzeit beim *FollowSM*-System umsetzbar, weil keinerlei spezielle Funktionalität benötigt wird. Allerdings führt es dazu, daß initial *alle* Verhaltenspfade für die verschiedenen Parameterwerte parallel untersucht werden. Dies ist im allgemeinen ineffizienter als der zusätzlich verwirklichte Ansatz über eine besondere Parameter-Ermittlungsprozedur, in der protokollglobales Wissen über die Auswirkungen des Parameters ausgenutzt werden kann.

Ein Sonderfall für die Parameterschätzung ist die Schätzung der tatsächlichen Zeitgeberlaufzeiten, die die IUT verwendet. Denn diese Zeiten sind anhand der Beobachtung nicht exakt meßbar und bereits im OTEFSM-Modell mit Unsicherheit ausdrückenden Zeitintervallen dargestellt (vgl. Abschnitt 3.2.1). Der geschilderte Alternativansatz über eine normale Zustandsvariable ist zur Ermittlung der „tatsächlichen" Zeitgeberlaufzeiten daher ungeeignet. Auf der anderen Seite wird in relevanten Protokollen bei diesen Zeiten besonders häufig auf verbindliche numerische Festlegungen im Protokollstandard verzichtet, um den Einsatzspielraum nicht einzuengen. Man denke z. B. an den Vergleich einer Datenübertragung über einige Meter Glasfaserleitung mit einer Satellitenfunkstrecke. Bei letzterer führt die Signallaufzeit zu minimalen Antwortzeiten in der Größenordnung einer halben Sekunde.

Jedes zeitgeberabhängige Schaltereignis liefert eine minimale und eine maximale tatsächliche Zeitgeberlaufzeit, $t_{\min}$ und $t_{\max}$. Dazu werden zunächst die früheste und späteste Ausführungszeit t_0, t_1 der Transition, die den verantwortlichen Zeitgeber gestartet hat, und die früheste und späteste Schaltzeit t'_0, t'_1 der ausgelösten zeitabhängigen Transition ermittelt. Außerdem sind die von der startenden Transition eingestellte Minimal- und Maximallaufzeit

θ_0, θ_1 bekannt. Somit gilt:

$$\begin{aligned} t_{\min} &= \max\{t'_0 - t_1; \theta_0\} \\ t_{\max} &= \min\{t'_1 - t_0; \theta_1\} \end{aligned}$$

Figur A.22 verdeutlicht die Herkunft dieser Werte noch einmal in grafischer Form.

Mit Hilfe dieser Paare $(t^i_{\min}, t^i_{\max})$ ermittelt der *FollowSM*-Analysator einige statistische Kennzahlen, um die tatsächliche Zeitgeberlaufzeit in der IUT zu charakterisieren. Im folgenden steht k für die Anzahl der beobachteten Zeitgeberabläufe:

- Umfang k der beobachteten Stichprobe.
- Globales Minimum t_{Min} der vorkommenden Zeiten:

$$t_{\text{Min}} = \min_{i=1..k} t^i_{\min}$$

- Globales Maximum t_{Max} der vorkommenden Zeiten, analog.
- Standardabweichung $\sigma t_{\min}$ der minimalen Laufzeiten:

$$\sigma t_{\min} = \sqrt{\frac{1}{k}\sum_{i=1}^{k}\left(t^i_{\min} - \frac{1}{k}\sum_{j=1}^{k} t^j_{\min}\right)^2}$$

- Standardabweichung $\sigma t_{\max}$ der maximalen Laufzeiten, analog.
- Mittelwert $\bar{t}$ als Schätzwert der Laufzeit über alle Intervalle:

$$\bar{t} = \frac{1}{2k}\sum_{i=1}^{k} t^i_{\max} + t^i_{\min} = \frac{\overline{t_{\max}} + \overline{t_{\min}}}{2}$$

- Mittlere Breite Δt der rekonstruierten Laufzeitintervalle:

$$\Delta t = \frac{1}{k}\sum_{i=1}^{k} t^i_{\max} - t^i_{\min} = \overline{t_{\max}} - \overline{t_{\min}}$$

Diese Kennzahlen vermitteln einen umfassenden Eindruck vom Zustandekommen der Laufzeitschätzung $\bar{t}$. So zeigt z. B. ein Vergleich der Standardabweichungen der Minimal- bzw. Maximalzeiten mit der mittleren Intervallbreite, ob eher eine ungenaue Schätzung vorliegt (breite Intervalle) oder die tatsächliche Laufzeit stark schwankt (hohe Standardabweichung).

3.3.7 Arbeitsersparnis für den Anwender

Der mögliche Nutzen der nachgeschalteten Analysen bei der Berichterstellung läßt sich abschätzen, wenn man die exemplarischen Berichtslängen in Figur A.19 betrachtet: Obwohl die Auflistung aller Suchschritte für den Anwender wenig brauchbar ist, stellt sie dennoch eine Beschreibung genau derjenigen „Überlegungen" dar, die auch ein menschlicher Protokollexperte anstellen müßte, um mit vergleichbarer Zuverlässigkeit eine Aussage über die Protokollkonformität der IUT treffen zu können. Derjenige Analysebericht, der nur noch die beobachteten Fehlerklassen mit statistischer Ursacheneingrenzung beinhaltet, stellt dagegen eine sehr kompakte und vollständige Charakterisierung des IUT-Verhaltens dar und ist um zwei Größenordnungen kürzer. Dieser Größenvergleich liefert damit einen brauchbaren Anhaltspunkt für die Aufwandsersparnis, die das *FollowSM*-Verfahren gegenüber der manuellen Interpretation der Ausgaben eines konventionellen Protokollmonitors erzielt.

3.3.8 Implementierung

Der *FollowSM*-Prototyp wurde in C++ implementiert, um einen objektorientierten Entwurf – mit generischen Datentypen, um protokollunabhängige Konzepte mit protokollspezifischen Einzelheiten instantiieren zu können – mit zufriedenstellender Effizienz und guter Portierbarkeit auf alle üblichen Plattformen zu kombinieren. Da der GNU-C-Compiler *gcc* verwendet wurde, ist eine unmittelbare Verwendung der Software auf allen Systemen sichergestellt, wo der *gcc* verfügbar ist. Eine einfache grafische Benutzerschnittstelle für das X-Window-System ist als separates Programm in Tcl/Tk realisiert. Figur A.23 zeigt ein Beispiel für die Bildschirmanzeige der Oberfläche.

Datenmodell

Figur A.24 gibt einen Überblick über das Datenmodell, auf dem der objektorientierte Programmentwurf basiert. Die Darstellung lehnt sich an an Entitäten-Relationen-Diagramme aus der Datenbankmodellierung (nach [Che76]). Die Rechtecke repräsentieren Klassen, während die Rauten Relationen beschreiben. Punkte an Rauten markieren die n-Seite einer $1 : n$-Beziehung, Pfeile symbolisieren Ableitung (Vererbung), in Pfeilrichtung gelesen als „ist abgeleitet von". Die Namen der protokollspezifischen Klassen (gestrichelte Rechtecke) sind mit *XXX* als Platzhalter für die Protokoll- bzw. Schichtbezeichnungen versehen. Alle Klassen mit durchgezogenen Rechtecken bilden den protokollunabhängigen „Bibliotheks"-Anteil des *FollowSM*-Systems.

Man erkennt, daß die Zustandsbeschreibungen (Klasse *State*) nicht nur modulweise, sondern auch schichtweise organisiert sind (Klassen *Module* und *Layer*). Die PDUs (Klasse *PDU*) werden über eine spezielle Speicherstruktur mit Hilfe der Klasse *Forget* verwaltet. Sie werden gelesen, wann immer die *Entwickle*-Prozedur eine neue PDU anfordert, jedoch keine PDU für den gerade bearbeiteten Modultyp im Zwischenspeicher (Klasse *PduCache*) mehr vorhanden ist. Dabei können – quasi zwischendurch – auch PDUs zu und von anderen Modulen auftreten, die nebenbei dekodiert und im Zwischenspeicher abgelegt werden müssen. Trotzdem dient der

Zwischenspeicher nur dem Ausgleich von Ausgabeverzögerungen und solchen Verschränkungen zwischen den PDU-Strömen verschiedener Module und Schichten. Wenn der Analysefortschritt *aller* Module einen Zeitpunkt überschritten hat, kann der Zwischenspeicher alle älteren PDUs bereits wieder vergessen. Daher stammt der Name der Speicherverwaltungsklasse *Forget*. Wenn mehrere Schichten gleichzeitig analysiert werden, müssen die PDUs einer höheren Schicht aus den Datenfeldern der Schicht darunter extrahiert werden. Diese Aufgabe hat bei *FollowSM* der Dekoder der unteren Schicht zu erfüllen, der zu diesem Zweck rudimentäre Protokollfunktionalität enthalten muß, um beispielsweise Paketwiederholungen von erstmals gesendeten Daten zu unterscheiden oder um die korrekte Nachrichtensequenz für die höhere Schicht wiederherzustellen. Die OTEFSM-Module können diese Aufgabe nicht erfüllen, weil ihre Zustandsübergänge im allgemeinen unsicher, d. h. spekulativ ausgeführt werden. Figur A.25 zeigt die Aufrufrelation der an der PDU-Verwaltung innerhalb eines Protokollstapels beteiligten Klassen und Methoden. Der dargestellte Stapel umfaßt die Protokolle Q.2110 und Q.2931 als Schichten 2 und 3 der Signalisierung am UNI (*user-network interface*, Teilnehmer-Netz-Schnittstelle) des B-ISDN, soll aber gleichermaßen als Beispiel für den allgemeinen Fall dienen. Im Diagramm sind die Aufrufeingänge der Methoden mit einem schwarzen Dreieck gekennzeichnet, die Ausgänge befinden sich jeweils gegenüber. Waagerechte Linien direkt am Ein- oder Ausgang einer Methode werden nur von dieser Methode selbst verwendet.

Transitionen

Jede Transition der OTEFSM wird bei *FollowSM* als ein Satz von Methoden der zugehörigen Zustandsklasse (*StateXXX* im Datenmodell) beschrieben. Diese Methoden sind:

- ***fPre(ipdu)*** testet die *Vorbedingung* der Transition, das sind die Zustandsbedingung und die Anforderung an eine evtl. erforderliche Eingabe-PDU *ipdu*. Zu diesem Test gehört die Berechnung derjenigen Zustandsteilmenge, für die die Vorbedingung erfüllt ist.

- ***fNPre(ipdu)*** bildet die Negation der Vorbedingung. Diese scheinbare Redundanz rührt daher, daß für unsicherheitsbehaftete Zustandsbeschreibungen *ja* und *nein* durchaus gleichzeitig gelten kann, wenn auch nicht für dieselben einzelnen Elementarzustände (vgl. Abschnitt 3.3.1). Außerdem können unterschiedliche Einschränkungen für die Repräsentierbarkeit der Ja- und der Nein-Fälle gelten.

- ***fPost(ipdu,opdu)*** ist die *Nachbedingung* der Transition, so genannt, weil sie die Forderungen an eine Ausgabe-PDU *opdu* der Transition untersucht, *nachdem* die Transition als schaltfähig bzw. schaltpflichtig festgestellt wurde. Sie wird nicht als Negation benötigt, weil ihr Ergebnis keinen Einfluß auf alternative Transitionen hat, wie das bei der Vorbedingung der Fall ist.

- ***fTrans(ipdu,opdu,t0,t1)*** ist die Zustandsüberführungsfunktion der Transition, wenn sie zwischen *t0* und *t1* schaltet und dabei die Eingabe *ipdu* konsumiert, die Ausgabe *opdu* produziert.

Die Zustandsänderungen sind auch bei den Bedingungen als direkte Änderung des Zustandsobjekts implementiert, für das die Methoden aufgerufen werden. Aus diesem Grund führen

sämtliche Operationen auf Zustandsbeschreibungen über ein Kennzeichen darüber Buch, ob eine Änderung der Zustandsbeschreibung aufgetreten ist, damit sich der Aufrufer um die Aufbewahrung und Wiederherstellung noch benötigter Zustandsinformationen kümmern kann.

Um ein Gefühl für die praktische Realisierung der Transitionsdefinitionen zu geben, zeigt Figur A.26 ein Beispiel für den C++-Programmtext einer Transition aus dem Q.2110-Protokoll. Die vereinfachte Notation der Methoden und der logischen Operatoren wird durch den Einsatz von Präprozessormakros erreicht.

3.4 Spuranalyse anhand von SDL-Spezifikationen

Die bishereigen Abschnitte dieses Kapitels sind nicht auf die Frage eingegangen, wie die OTEFSM-Spezifikationen zur Spuranalyse aus einer existierenden formalen Spezifikation des Zielprotokolls abgeleitet werden kann (vgl. Abschnitt 3.2). Dieser Schritt ist Gegenstand dieses Abschnitts, der eine Zusatzkomponente in Form eines Übersetzers beschreibt, der eine OTEFSM-Spezifikation in der in Figur A.26 angedeuteten Form aus einer Protokollspezifikation in der Sprache SDL (vgl. Abschnitt 2.1.2) generiert.

Die in diesem Abschnitt beschriebene Lösung wurde in [Pil97] ausgearbeitet.

3.4.1 Motivation

Anhand von Transitionsdefinitionen wie in Figur A.26 sieht man leicht ein, daß der manuelle Entwurf einer OTEFSM zur Spuranalyse sowohl aufwendig als auch fehleranfällig ist. Der Entwurf einer OTEFSM-Spezifikation für das Abschnittssicherungsprotokoll der Breitband-ISDN-Signalisierung, SSCOP nach ITU Q.2110 [IT94a], auf Grundlage der SDL-Spezifikation im Standarddokument erforderte ungefähr einen Personenmonat an Entwicklungsarbeit. Eine Verwendung von SDL-Spezifikationen, um OTEFSM-Spezifikationen für *FollowSM* automatisch zu generieren, ist daher ein logischer und von der Idee formaler Spezifikationstechniken her geradezu zwingender Weg, um die Korrektheit konkreter *FollowSM*-Analysatoren sicherzustellen und ihre Entwicklungszeit zu minimieren.

3.4.2 Problemanalyse

Zunächst soll untersucht werden, ob und inwiefern ein vollständiger und korrekter *FollowSM*-Analysator aus einer SDL-Spezifikation abgeleitet werden kann.

Berechenbarkeit der Spuranalyse

Leider ist es, wie in Abschnitt 3.1.2 gezeigt wurde, nicht möglich, die Spuranalyse für beliebige immerhin terminierende SDL-Spezifikationen zu berechnen. Aus Theorem 2 folgt daher, daß die automatische Ableitung einer äquivalenten OTEFSM aus einer SDL-Spezifikation nicht im allgemeinen berechenbar ist. Die technische Erklärung hierfür besteht darin, daß gewisse SDL-Konstrukte im OTEFSM-Kalkül nicht modellierbar sind.

Besonderheiten der OTEFSM

Aufgrund der in Abschnitt 2.2.1 geschilderten besonderen Anforderungen an einen realen Spuranalysator verfügt die OTEFSM über einige Modellierungsaspekte, die über den Modellierungsumfang von SDL hinausgehen. Diese Aspekte, die vor allem Zeitbedingungen betreffen, lassen sich nicht aus einer SDL-Spezifikation gewinnen.

Figur A.27 stellt die Relation der mit SDL bzw. einer Anzahl paralleler OTEFSMs modellierbaren Mengen von Systemen grafisch dar. Tabelle 3.3 nennt einige Aspekte der Modellierung, die eines der Kalküle dem jeweils anderen voraushat.

Informelle Anteile

In vielen praktisch verfügbaren Protokollstandards sind SDL-Spezifikationen mit informellen Anteilen vorhanden. So kommt z. B. in der SDL-Spezifikation zum Protokoll Q,2931 der ITU [IT94b] wiederholt die „implementierungsabhängige Prozedur" *Check Message* vor, deren Aufgabe nur im natürlichsprachlichen Teil des Standards näher beschrieben wird. Weitere Details werden nur als Kommentare in der Spezifikation vermerkt. Zwar stellen derart unvollständige Spezifikationen eine Entscheidung des SDL-Anwenders dar und sind keineswegs in der Ausdrucksfähigkeit von SDL begründet. In der Praxis ist aber mit solchen Spezifikationen zu rechnen, die eine vollständig formale Ableitung einer OTEFSM von vornherein ausschließen.

3.4.3 Lösungsansatz

Aus diesen Gründen wurde ein Übersetzer entworfen, der aus einer SDL-Spezifikation, für die gewisse Einschränkungen hinsichtlich des verwendeten Sprachumfangs gelten, ein *Gerüst* für die OTEFSM-Spezifikation des Spuranalysators erzeugt. Die in Abschnitt 3.4.2 genannten Lücken müssen zwangsläufig durch eine anschließende manuelle Nachbearbeitung dieses Gerüstes geschlossen werden. Trotzdem bedeutet der SDL-Übersetzer eine erhebliche Arbeitsersparnis, weil sich die Struktur der OTEFSM-Spezifikation direkt aus den SDL-Transitionen ableiten läßt. Der zeilenmäßig umfangreichste und vor allem monotonste Teil des OTEFSM-Entwurfs bleibt dem Entwickler erspart.

Einschränkungen für die Spezifikation

Eingabe des Übersetzers ist eine Systemspezifikation in SDL/PR. Die folgende Auflistung nennt die wesentlichen SDL-Konstrukte, die bei der OTEFSM-Ableitung *nicht* behandelt werden können:

- Explizite Prozeßerzeugung mit *Create*. OTEFSM-Module werden implizit erzeugt, sobald zugehörige PDUs auftreten. Explizite Prozeßerzeugung würde gegen die Unabhängigkeitsprämisse (vgl. Abschnitt 3.3.4) verstoßen.
- Zugriff auf den Zustand anderer Prozesse mit *View*-Ausdrücken sowie exportierten und importierten Variablen. Auch dies läßt die Unabhängigkeitsprämisse nicht zu.

- *Remote-Procedure-Input*-Transitionen, bei denen eine Prozedur in einem anderen Prozeß aufgerufen wird, verbieten sich aus demselben Grund.
- *Save*-Operationen, bei denen bestimmte Eingabesignaltypen zur späteren Verarbeitung aufbewahrt werden, um zunächst andere Signale in der Warteschlange behandeln zu können, sind im OTEFSM-Modell nicht darstellbar. Dort müssen alle möglichen Eingaben jederzeit verarbeitet werden können, was zu Änderungen im Transitionsentwurf führt.
- In SDL können Prozesse in mehrere *Service*-Bereiche mit interner Kommunikation aufgeteilt werden. Dies widerspricht dem Ein-Automaten-Paradigma der OTEFSM mit vollständig beobachtbaren Ein- und Ausgaben.
- Direkt oder indirekt rekursive Prozeduren sind nicht zulässig, weil Prozeduraufrufe wegen der Besonderheiten der unsicherheitsbehafteten Operations- und Prädikatauswertung durch Expansion des Prozedurrumpfes übersetzt werden.
- Die übersetzbaren arithmetischen Operationen sind durch die vorhandenen Operationen auf Unsicherheit modellierenden Datentypen begrenzt (vgl. Abschnitt 3.3.1). Hier sind bei Bedarf Erweiterungen möglich.

Im allgemeinsten Fall kann eine kommunizierende Instanz in SDL so mit Hilfe mehrerer SDL-Prozesse spezifiziert sein, daß eine Übertragung in das OTEFSM-Modell für *FollowSM* einen neuen Entwurfsprozeß erfordert. Dieser ist aus Berechenbarkeitsgründen nicht automatisierbar. Als praktische Lösung wird innerhalb der SDL-Spezifikation ein Prozeßtyp gesucht, der für die Abwicklung der Kommunikation hinsichtlich eines in sich abgeschlossenen PDU-Stroms verantwortlich ist und daher einen OTEFSM-Modultyp für den *FollowSM*-Analysator liefert. Diese Lösung führt direkt zum Erfolg, wenn die Spezifikation das Prozeßkonzept von SDL benutzt, um die PDUs modellierenden SDL-Signale von einem zentralen Verteilerprozeß V erzeugen und empfangen zu lassen, der sie weitgehend unverändert an diejenigen SDL-Prozesse P_i weiterleitet, die für die einzelnen virtuellen Verbindungen zuständig sind. In diesem Szenario werden die P_i zu OTEFSM-Modulen, während vom Prozeß V vollständig abstrahiert werden kann. Dieser Fall gilt in Reinform beispielsweise für das ITU-Protokoll Q.2931. Die Prozeßstruktur der SDL-Spezifikation dieses Protokolls ist in Figur A.28 wiedergegeben. Die grau unterlegten Prozesse werden zu OTEFSM-Modulen des Analysators.

Für die Information, welche Signale PDUs repräsentieren und welche Prozesse zu OTEFSM-Modulen zu übersetzen sind, erwartet der Übersetzer die folgenden drei Signallistendeklarationen in der Spezifikation. In Bezug auf die Prozesse ist diese Deklaration nicht semantisch korrekt im Sinne von SDL, was bei dieser Spezialanwendung aber nicht stört:

```
system <name>;
   :
   signallist PROCESSES = <name>,...,<name>;
   signallist INPDU     = <name>,...,<name>;
   signallist OUTPDU    = <name>,...,<name>;
```

```
  ⋮
endsystem;
```

Übersetzung der Transitionen

Nach Maßgabe der Semantik von SDL bzw. des OTEFSM-Modells können die SDL-Transitionen direkt in OTEFSM-Transitionen überführt werden. Allerdings gelten strengere Regeln für den Aufbau von OTEFSM-Transitionen, so daß eine SDL-Transition häufig in mehrere OTEFSM-Transitionen zerlegt werden muß. Die strengeren Regeln für OTEFSM-Transitionen sind insbesondere:

- Höchstens eine Ein- und Ausgabe pro Transition.
- Höchstens ein Zeitgeber für die Auslösung einer zeitabhängigen Transition.
- Zustandsabhängige Bedingungen müssen in der *Vorbedingung* der Transition zusammengefaßt sein, in der Zustandsüberführungsfunktion dürfen sie nicht vorkommen.
- Keine Sprunganweisungen und Marken.

Im folgenden werden die erforderlichen Transformationsregeln für SDL-Transitionen mit Hilfe von SDL/GR-Diagrammen dargestellt. Resultierende SDL-Transitionen, die die obigen Regeln erfüllen, lassen sich anschließend unmittelbar in OTEFSM-Transitionen umformen.

Transitionen mit *mehreren Stimuli*, d. h. mehr als einer auslösenden Eingabe-PDU oder mehr als einem auslösenden Zeitgeber, werden in je eine eigene Transition pro Stimulus umgewandelt (Regel R1, siehe Figur A.29).

Transitionen mit *mehreren Ausgaben* müssen in ähnlicher Weise expandiert werden. Weil die dabei erzeugten neuen Transitionen nicht nebeneinander betrachtet, sondern *nacheinander* ausgeführt werden müssen, ist es erforderlich, neue Zustände einzuführen, die die Transitionen verketten. Im OTEFSM-Modul werden die Folgetransitionen als *unmittelbare Transitionen* realisiert, die vom neuen Zwischenzustand aus unbedingt schalten (Regel R2, siehe Figur A.30).

Verzweigungen in Transitionen müssen in die Vorbedingung vorgezogen werden, woraus pro Verzweigungsast eine eigene Transition resultiert (Regel R3, siehe Figur A.31). Dabei müssen zusätzlich die Bedingungen angepaßt werden, damit sich die Semantik nicht ändert, wenn die Transition vor der Verzweigung Änderungen an einer Variablen vornimmt, die in der Verzweigungsbedingung vorkommt. Figur A.32 verdeutlicht dies an einem Beispiel.

Sprünge zu Sprungmarken müssen durch den Transitionstext am Sprungziel ersetzt werden, da es keine Sprünge in OTEFSMs gibt (Regel R4, siehe Figur A.33).

Der Fall, daß eine *Verzweigung* hinter einer *Marke* auftritt, wird gesondert behandelt, um einerseits die Anzahl der erzeugten OTEFSM-Transitionen zu reduzieren und andererseits Schleifenkonstruktionen zu erlauben, bei denen eine reine Expansion der Transitionen nach den Regeln R3 und R4 nicht terminieren würde (Regel R5, siehe Figur A.34). Das Problem der Nichttermination tritt auf, wenn hier für M1 und M2 dieselbe Marke auftritt. Dieser Fall einer Schleife ist beispielhaft in Figur A.35 dargestellt.

Bei der Übersetzung der Variablen der SDL-Spezifikation muß der Übersetzer entscheiden, in welchen Fällen unsicherheitsbehaftete Zustandsvariablen der OTEFSM resultieren und wann es sich nur um Hilfsvariablen der Zustandsüberführung handelt. Solche Hilfsvariablen werden direkt in C++-Variablen übersetzt, da jede unnötige Vergrößerung der unsicherheitsbehafteten Zustandsbeschreibung einen Mehraufwand bei den zu speichernden Zustandsobjekten verursacht. Außerdem führt eine Aufnahme in die Zustandsbeschreibung überflüssigerweise zur Anwendung der komplizierten Auswertungsregeln für logische Ausdrücke (vgl. Abschnitt 3.3.1) sowie zur Anwendung der Transformationsregel R3, sobald die Hilfsvariable in einer Bedingung vorkommt. Das Kriterium für eine nicht zustandsrelevante Hilfvariable h ist, daß in *jeder* Transition, in der h vorkommt, das erste Auftreten von h ein schreibender Zugriff ist, also eine Zuweisung an h.

Manuelle Nachbearbeitung

Die bereits erörterten Lücken in der automatischen Übersetzung müssen durch eine anschließende manuelle Nachbearbeitung des OTEFSM-Programmtextes geschlossen werden. Darunter können folgende Arbeiten fallen:

- Strukturelle Änderungen, um mehrere Prozesse der Spezifikation zusammenfassen zu können, wenn keine 1:1-Beziehung zwischen SDL-Prozessen und nötigen OTEFSM-Modulen herstellbar ist.
- Festlegung der Repräsentationsräume der Zustandsvariablen durch Wahl der entsprechenden Datentypen.
- Ergänzung der konkreten Zeitgeberzeiten und Toleranzintervalle.
- Realisierung von Kompromißlösungen unter Verzicht auf Vollständigkeit aus Effizienzgründen oder wenn die Operationen in der SDL-Spezifikation nicht mit Hilfe der vorhandenen Unsicherheit modellierenden Datentypen nachbildbar sind.

Da die konkrete Transfersyntax der PDUs nicht in SDL beschrieben werden kann, wird außerdem das Dekoder-Modul des Analysators manuell entworfen.

Trotz der erforderlichen manuellen Nachbearbeitung reduziert sich der Gesamtaufwand für die Entwicklung eines protokollspezifischen *FollowSM*-Analysators dank des SDL-Übersetzers größenordnungsmäßig auf ein Viertel der Zeit für den komplett manuellen Entwurf.

3.4.4 Implementierung

Der Übersetzer ist in C implementiert. Dabei wurden außerdem ein im Rahmen des *SITE*-Projekts an der Berliner Humboldt-Universität entwickelter SDL-Parser [Lut93] und das an der Universität Twente entwickelte „Meta-Tool" *Kimwitu* [vE+93] verwendet, das die Konstruktion und Manipulation von Baumstrukturen anhand von Quelltext komfortabel ermöglicht.

3.5 Praktische Anwendung

Dank einer Zusammenarbeit mit GMD-Fokus ergab sich die Möglichkeit, den *FollowSM*-Analysator im „Online"-Betrieb einzusetzen.

3.5.1 Versuchsaufbau

Figur A.36 illustriert den Versuchsaufbau. Der zu messende Verkehr entsteht durch die ausführbare SSCOP-Testsuite (*executable testsuite*, ETS) auf dem Protokolltester K1297 von Siemens. In der ETS werden der IUT sowohl protokollkonformes Verhalten präsentiert als auch, um die Fehlerbehandlungsprozeduren zu testen, Fälle mit Protokollverstößen. Damit bietet dieser Meßaufbau einen wesentlich höheren Überdeckungsgrad im Hinblick auf die verifizierbaren Protokollprozeduren als irgendeine reale Kommunikationsverbindung.

Die Gegenseite bildet eine SSCOP-Implementierung. Hier standen ein Switch von FORE sowie ein prototypischer Protokollstapel von CSELT zur Verfügung. Das eingezeichnete Kanal-Multiplexing per Umsetzung der VPCIs (*virtual path connection identifier*, Bezeichner der virtuellen Pfadverbindung) ist erforderlich, weil die auf dem Arbeitsplatzrechner eingesetzte ATM-Karte (SBA200 von FORE) über nur einen Eingang verfügt.

3.5.2 Meßsituationen

Da der *FollowSM*-Monitor zu jeder Zeit nur das korrekte Ein-/Ausgabeverhalten *einer* der beteiligten Instanzen untersucht, ergibt der Versuchsaufbau zwei grundsätzlich unterschiedliche Meßsituationen:

1. *FollowSM* untersucht das Verhalten des K1297. In diesem Fall sollte der Monitor genau die von der ETS injizierten Fehler als Protokollverstöße melden, während die Testfälle mit protokollkonformem Verhalten als korrekt beurteilt werden sollten.

2. *FollowSM* untersucht das Verhalten der SSCOP-Implementierung. In diesem Fall führen die injizierten Fehler *nicht* zu Fehlermeldungen von *FollowSM*. Statt dessen prüft der Monitor die protokollkonforme Fehlerbehandlung seitens der Implementierung. Sofern die Implementierung also protokollkonform arbeitet, sollte *FollowSM* keinerlei Fehler melden.

   Außerdem kann das *FollowSM*-Resultat mit den Ergebnissen der Testsuite für die einzelnen Testfälle verglichen werden: Wenn die Implementierung fehlerhaft ist, sollten genau die von der Testsuite festgestellten Verstöße auch von *FollowSM* erkannt werden.

Im Hinblick auf eine sinnvolle Einsetzbarkeit im „Online"-Betrieb sind die Rechenzeiten für die Fehleranalyse von besonderem Interesse. Daher wurde der Monitor um die Funktion erweitert, den Anteil der inaktiven Wartezeit des Monitorprozesses auf neue PDUs an der gesamten Meßdauer zu ermitteln.

3.5.3 Ergebnisse

Die Ergebnisse der durchgeführten Experimente werden hier gegliedert in die Ergebnisse bei der Erkennung von Protokollfehlern und die Ergebnis in Bezug auf die Rechenzeit.

Resultate der Messungen

Herausragend zu erwähnen ist die Messung einer Testfallsequenz mit 186 PDUs und ca. 34 s Dauer, in der die ETS zwei Fälle mit *„fail"* bewertete, *FollowSM* jedoch keinerlei Protokollverstoß erkannte. Eine manuelle Analyse ergab, daß das vorliegende Verhalten in der Tat protokollkonform ist, die ETS in einem relativ komplizierten Fall jedoch eine falsche Akzeptanzbedingung für einen PDU-Parameter formuliert. Dieser Versuch hat also einen Fehler in der ETS aufgedeckt.

Zeitverhalten

Der Zeitaufwand des Analyse-Algorithmus blieb während der Experimente stets unkritisch für den „Online"-Betrieb. Tabelle 3.4 zeigt drei Beispiele für die Rechenlast der verwendeten SPARCstation 10 beim „Online"-Monitoring, gemessen am Wartezeit-Meßzeit-Verhältnis. Unter *CPU-Zeit* ist außerdem die reine Prozeß-Rechenzeit bei einer „Offline"-Analyse der PDU-Sequenzen auf einer SPARCstation 5 vermerkt.

Man sieht deutlich, wie die bei steigender Fehlerzahl erforderlichen Zustandsrekonstruktionen die Rechenzeit verlängern. Aber selbst der ungünstige Fall 3, wo gerade zum Erreichen einer hohen Fehlerzahl nicht die Implementierung, sondern der Tester „überprüft" wird, erzeugt keine nennenswerte Belastung des Rechners.

Dennoch ist dieses Ergebnis relativiert zu sehen, denn es rührt auch daher, daß der Tester bei der Ausführung der Testsuite sehr langsam ist und daß es in den Testfällen zwangsläufig auch Wartezeiten auf Zeitgeber gibt. Praktisch läßt sich eine Daumenregel von etwa 200 bis 400 analysierten PDUs pro Sekunde, je nach Fehlergehalt der Kommunikation, für die eingesetzte Hardware ablesen.

SDL	OTEFSM
Abhängigkeit zwischen den Zuständen mehrerer Prozesse	Toleranzbereich für die Beobachtungs-zeitpunkte der Ausgabenachrichten
Explizite Prozeßerzeugung und -terminierung	Zeitgeber mit Ablaufintervallen statt -zeitpunkten
beliebig viele Ein- und Ausgabekanäle des Gesamtsystems	Kompromißentscheidung hinsichtlich der Modellierung von Unsicherheit

Tabelle 3.3: Vergleich von SDL und dem OTEFSM-Modell anhand von Systemaspekten, die nur in einem der beiden Formalismen darstellbar sind.

Nr.	überwachte Instanz	PDUs	Meßdauer [s]	Fehler	inaktiv [%]	CPU-Zeit, Sparc5 [s]
1	CSELT	187	70	0	99,0	0,44
2	FORE	215	30	2	97,8	0,50
3	1297	187	60	56	96,9	1,10

Tabelle 3.4: Rechnerbelastung beim „Online"-Monitoring.

Kapitel 4

Protokoll-Lernen

Im vorangegangenen Kapitel wurde ein Verfahren vorgestellt, das auf der Grundlage einer *bekannten* Protokollspezifikation ein im Rahmen des Einsatzkontextes möglichst gutes Ergebnis bei der Analyse einer zu einem beliebigen Zeitpunkt beobachteten Spur der Protokollabwicklung erzielt. Es wurde deutlich, daß eine voll automatisierte Ableitung des Spuranalysators aus einer üblichen Protokollspezifikation z. B. in SDL nicht erfolgen kann. Daher verursacht die Erzeugung eines konkreten Spuranalysators für ein neues Zielprotokoll einen gewissen Aufwand:

1. Es muß eine vollständig formale, d. h. nicht mit semantisch relevanten Kommentaren versehene SDL-Spezifikation des Zielprotokolls vorliegen. Wenn mehrere Protokollversionen oder nur eine bestimmte Protokollversion mit genau festgelegten PICS-Parametern auf dem beobachteten Kommunikationskanal ablaufen dürfen, müssen alle diese Versionen bzw. die exakte Parametrisierung in der Spezifikation beschrieben sein.

2. Aufgrund der geschilderten Berechenbarkeitsproblematik und der beschriebenen Zusatzanforderungen der *passiven* Spuranalyse ist eine zusätzliche manuelle Bearbeitung des OTEFSM-Modells erforderlich, das den Analysator spezifiziert. Der Umfang und die Fehleranfälligkeit dieser Nachbearbeitung hängen vom Charakter des Zielprotokolls und seiner Spezifikation in SDL ab. Schlimmstenfalls ist hier noch echte Entwicklungsarbeit zu leisten, gefolgt von den dann erforderlichen Tests der resultierenden OTEFSM-Spezifikation („Testen des Testers").

Um diesen Aufwand bei der Analysatorerzeugung vollständig zu eliminieren, wird in diesem Kapitel der zweite Ansatz dieser Arbeit entwickelt: Ein Lernverfahren, das den Analysator nicht aus einer allgemeinen Beschreibung des Protokolls ableitet, sondern aus Beispielen der korrekten Abwicklung des Zielprotokolls. Die vorliegende Lernaufgabe und der Einsatzkontext eines solchen Systems wurden bereits in Abschnitt 2.2.2 informell dargestellt.

Im nächsten Abschnitt erfolgen zunächst die Formalisierung der Lernaufgabe sowie eine Charakterisierung des Problems und der möglichen Lösungen. Anschließend werden die schon in Abschnitt 2.2.2 dargestellten zwei Lernphasen und die Anwendungsphase einzeln untersucht und Algorithmen dafür entwickelt. Das Kapitel schließt mit einer Darstellung erster Ergebnisse aus der praktischen Anwendung des lernenden Spuranalysators.

4.1 Theoretische Fundierung

In diesem Abschnitt werden die Eingabe und Ausgabe des Lernverfahrens formal spezifiziert und die resultierende Lernaufgabe hinsichtlich ihrer Lösbarkeit und der möglichen Qualität des Ergebnisses untersucht.

4.1.1 Formalisierung der Lernaufgabe

Bereits in Abschnitt 2.2.2 wurde dargestellt, daß das Lernen von Protokollregeln aus Nachrichten, die völlig undekodiert in Form von Bitketten vorliegen, nicht machbar erscheint. Grund dafür sind die starke Kontextabhängigkeit der Interpretation vieler Stellen dieser Bitketten und die ausgeprägt semantikorientierte Konstruktion der PDU-Syntax: Um die PDU-Struktur als Grundlage funktionsfähiger Protokollregeln automatisch abzuleiten, müßten bereits zahlreiche Aspekte der Protokollsemantik bekannt sein, die sich aber wiederum nur aus dem Zusammenspiel der Protokollregeln erschließt. Diesen sich selbst blockierenden Kreis quasi von außen zu durchbrechen, wäre eine große Herausforderung für zukünftige Untersuchungen.

Daher wird – im folgenden Unterabschnitt *Vorgabe* – eine Dekodierung des Zielprotokolls in einer Art Termsprache vorausgesetzt, die die Grundlage für das Lernverfahren bildet.

Aus ähnlichem Grund wird als Zielkonzept des Lernvorgangs kein erweiterter endlicher Automat in der üblichen Form verwendet, sondern, wie in Abschnitt 2.2.2 angedeutet, nur ein endlicher Automat mit Regeln über die Datenfelder einiger aufeinanderfolgender PDUs: Der datenerweiterte Protokollzustand ist das Kondensat des Protokollentwurfs. Seine Rekonstruktion aus einer Beobachtung ist im allgemeinen kein rein symbolischer Vorgang, sondern die Ableitung einer Entwurfsidee aus der Gesamtheit der gemachten Beobachtungen. Auch dies muß hier als interessantes Fernziel eingestuft bleiben.

Vorgabe

Vorgegeben sei die *abstrakte PDU-Syntax* ζ des Zielprotokolls als Tripel $\zeta = (\Sigma_i, \Sigma_o, \kappa)$:

- Σ_i ist eine endliche Menge von Symbolen, den *Typen von Eingabe-PDUs*.
- Σ_o ist eine endliche Menge von Symbolen, den *Typen von Ausgabe-PDUs*. Es gilt $\Sigma_i \cap \Sigma_o = \emptyset$, damit die Richtung einer PDU bereits mit dem Symbol dargestellt werden kann.
- $\kappa : \Sigma \rightarrow \mathbb{N}$ ist eine Funktion, die jedem PDU-Symbol die *Stelligkeit* seiner Attributierung zuordnet.

Im folgenden gilt die Abkürzung

$$\Sigma \stackrel{def}{=} \Sigma_i \cup \Sigma_o$$

für die Gesamtmenge der Symbole.

Die Bedeutung dieser Definition ergibt sich, sobald auf ihrer Grundlage die Menge aller PDUs $\Pi(\zeta)$ gemäß der Syntax ζ festgelegt ist:

$$\Pi(\zeta) = \Pi(\Sigma, \kappa) \stackrel{def}{=} \bigcup_{a \in \Sigma} \{a\} \times \mathbb{N}^{\kappa(a)} \tag{4.1}$$

Geschrieben wird eine PDU $\pi \in \Pi(\zeta)$ im folgenden in der Form $a(x_1, \ldots, x_{\kappa(a)})$.

Eingabe

Eingabe für den Lernalgorithmus sind eine Folge beobachteter PDUs, $p \in \Pi(\zeta)^*$, sowie eine Fenstergröße $w \in \mathbb{N} \setminus \{0\}$. Das Lehrbeispiel p muß als *fehlerfreies* Musterverhalten der IUT vorausgesetzt werden, da der Lernansatz ja davon ausgeht, daß die Regeln für korrektes Protokollverhalten nicht vorher bekannt sind, so daß

1. eventuelle fehlerhafte Verhaltensweisen der IUT innerhalb des Beispielablaufs w nicht erkannt und herausgefiltert werden können und
2. keinerlei Möglichkeit besteht, Fehlerbeispiele zusätzlich zu erzeugen, weil auch dies die Kenntnis der Protokollregeln voraussetzt.

Wenn das Lehrbeispiel dennoch – entgegen der gemachten Annahme – Fehlerfälle enthält, werden diese unvermeidlich als Teil des vermeintlich korrekten Sollverhaltens gelernt.

Die Fenstergröße w beschreibt, über wie viele aufeinanderfolgende PDUs Regeln gebildet werden sollen, die die korrekte Attributierung der PDUs mit Datenfeldinhalten wiederspiegeln (vgl. Abschnitt 2.2.2).

Ausgabe

Als Ausgabe des Lernvorgangs soll ein endlicher Automat A über dem Alphabet Σ erzeugt werden, dessen Zustände mit Datenfeldregeln attributiert sind. Formal ist dieser Automat ein Sextupel $A = (\Sigma, \kappa, w, S, \delta, \omega)$:

- Σ, κ, w sind die oben spezifizierte Beschreibung der PDU-Syntax und die Fenstergröße für die Datenfeldregeln.
- S ist die endliche Menge der *Zustände* des endlichen Automaten über den PDU-Typen.
- $\delta \subseteq S \times \Sigma \times S$ ist die Zustandsübergangsrelation des endlichen Automaten über den PDU-Typen.
- $\omega : S \to \mathbb{P}\, \Pi(\Sigma, \kappa)^w$ ist die Attributierung der Zustände des Automaten mit einem Prädikat, das die zulässigen Datenfeldinhalte der w Elemente langen PDU-Sequenzen

zu dem jeweiligen Automatenzustand angibt. Für diese Attributierung gilt folgende syntaktische Kontextbedingung:

$$\forall s \in S,\ (a_1(x_1^1,\ldots,x_1^{\kappa(a_1)}),\ldots,a_w(x_w^1,\ldots,x_w^{\kappa(a_w)})) \in \omega(s)$$
$$\exists (r_0,\ldots,r_{w-1}) \in S^w$$
$$(r_0,a_1,r_1),(r_1,a_1,r_2),\ldots,(r_{w-2},a_{w-1},r_{w-1}),(r_{w-1},a_w,s) \in \delta$$

D. h., daß die Knotenattributierungen im Zustandsgraphen des Automaten sich immer auf Transitionspfade beziehen, die im jeweiligen Knoten enden.

Im Rahmen der Lernaufgabe gibt es nun die über die obigen syntaktischen Bedingungen hinausgehende Forderung an A, daß A die Beobachtung $p = p_1, p_2, \ldots, p_l$ *vollständig* erklärt. Formal muß die folgende Aussage $\mathrm{bc}(A,p)$ erfüllt sein:

$$\begin{aligned}\mathrm{bc}(A,p) \stackrel{def}{=}\ & \exists (r_0, r_1, \ldots, r_l) \in S_A^l \\ & (r_0, a_{p_1}, r_1), (r_1, a_{p_2}, r_2), \ldots, (r_{l-1}, a_{p_l}, r_l) \in \delta_A \\ \wedge\ & \forall i \in w..l \quad (p_{i-w+1}, \ldots, p_i) \in \omega_A(r_i)\end{aligned} \tag{4.2}$$

Anwendung

In der Anwendungsphase der erlernten Protokollbeschreibung A wird eine neue Beobachtung p auf ihre Erklärbarkeit mit dem gelernten Automaten hin untersucht.

Gegeben: $p \in \Pi(\zeta)^*$.

Gesucht: Wahrheitswert der Aussage $\mathrm{bc}(A,p)$ (4.2).

4.1.2 Charakterisierung der Lernaufgabe

Nach der nun erfolgten Formalisierung der Lern- und Prüfaufgabe lassen sich Aussagen zur Lösbarkeit des Lernproblems und zur Eindeutigkeit der Lösung machen.

Weil keinerlei negative Beispiele zur Verfügung stehen, hat ein Lernalgorithmus bei der Lösung dieser Lernaufgabe eine sehr große Freiheit. Diese bewegt sich zwischen einer Protokollbeschreibung A_{min}, die ausschließlich die *beobachtete* PDU-Folge zuläßt, und einer Protokollbeschreibung A_{max}, die *alle* Verhaltensweisen im Rahmen der PDU-Syntax ζ akzeptiert. Die Figuren A.37 und A.38 stellen diese beiden Extreme grafisch dar für das Lehrbeispiel $a(17), b(17,42), c(59)$ und eine Fenstergröße $w = 2$. Beide Ausgaben sind in jeder Hinsicht korrekte Lösungen der Lernaufgabe!

Man erkennt daran, daß die Lernaufgabe für beide Teilprobleme, den endlichen Automaten der PDU-Typen und das Regellernen für die Datenfelder, nur heuristisch gelöst werden kann. Auch das Kriterium der minimalen Beschreibungslänge (*minimum description length*, MDL, z B. [KMU95]) für A führt nicht weiter, weil es in Abwesenheit von Negativbeispielen immer die unbrauchbare Antwort $A_{\max}$ erzwingt.

Daraus folgt, daß es eine einzige *richtige* Lösung dieser Aufgabe nicht geben kann. Bereits anhand der Aufgabenstellung läßt sich also ablesen, daß der Verzicht auf eine von außen vorgegebene Protokollspezifikation einen Kompromiß hinsichtlich der erreichbaren Qualität der

Spuranalyse erzwingt. Die mit dem Lernansatz im folgenden erzielten Ergebnisse müssen jederzeit relativ zu dieser grundlegenden Einschränkung bewertet werden. Es bleibt nichtsdestoweniger eine interessante Fragestellung, was innerhalb des nunmehr abgesteckten Rahmens erreichbar ist.

In Abschnitt 2.1.4 wurde bereits festgestellt, daß sich kein bekanntes Lernverfahren unmittelbar für das Problem des Protokollernens eignet. In den folgenden Abschnitten werden daher heuristische Lernalgorithmen für die beiden Phasen der Lernvorgangs entwickelt, die sehr speziell auf das vorliegende Problem zugeschnitten sind.

4.2 Algorithmus zum Regellernen

In diesem Abschnitt wird der Lernalgorithmus für die Phase des Regellernens über die PDU-Datenfelder vorgestellt. Dieser Lernalgorithmus ist zwar für den Protokollkontext entwickelt worden, kann von seiner Spezifikation her aber auch allgemein betrachtet werden. Er trägt den Namen *LARGE*-Algorithmus (für *learning arithmetical rules from good examples*, Lernen arithmetischer Regeln aus Positivbeispielen).

4.2.1 Spezifikation

Zunächst wird die elementare Lernaufgabe für den *LARGE*-Algorithmus beschrieben. Im Kontext des Protokollernens handelt es sich dabei um das Erlernen einer Regel für die Datenfelder *einer* ganz bestimmten Kombination von w PDU-Typen $a_i \in \Sigma$ an *einem* Zustand $s \in S$ des Automaten. Die Spezifikation der Ausgabe von *LARGE* umfaßt natürlicherweise die Definition der Hypothesensprache, die der Algorithmus verwendet.

Eingabe

Eingabe E für *LARGE* ist eine Menge von m Attributvektoren mit je n Komponenten, die Trainingsmenge:

$$E \subset \mathbb{N}^n \qquad |E| = m > 0 \ \wedge \ n > 0$$

Diese einfache Struktur der Eingabe wird dadurch ermöglicht, daß die Auswahl einer festen PDU-Sequenz entsprechend der Fenstergröße allen Attributen eine feste Position in den Beispielvektoren verleiht. Im obigen Fall mit dem Lehrbeispiel $a(17), b(17, 42), c(59)$ ergibt sich ein einziger Attributvektor: $E = \{(17, 17, 42, 59)\}$

Ausgabe

Schon in Abschnitt 2.1.4 wurde ausgeführt, daß arithmetische Zusammenhänge zwischen den Datenfeldinhalten benötigt werden, um korrektes Protokollverhalten zu charakterisieren. Um solche komplexen Regeln kompakt darstellen zu können, verwendet *LARGE* eine spezielle Form funktionsfreier Hornklausellogik, also DATALOG, als Hypothesensprache L.

Jede Hypothese $H \in L$ ist eine Menge von Klauseln

$$H = \{c_1; \ldots; c_k\}.$$

Jede Klausel c_i hat die Form

$$c_i = \mathrm{ok}(v_1, \ldots, v_n) \vdash \alpha_1, \ldots, \alpha_q, \beta_1, \ldots, \beta_p.$$

Dabei steht das ok-Prädikat für „Attributvektor ist regelkonform". Die Variablen $v_1, \ldots, v_n$ werden mit den Attributwerten eines Beispielvektors aus der Trainingsmenge E identifiziert. Für die Atome α_i $(1 \leq i \leq q)$ der Klausel gibt es folgende Möglichkeiten:

mul1:	$\mathrm{mul1}(v_{\mathrm{neu}}, v_j, K)$	$\Leftrightarrow$	$v_{\mathrm{neu}} = v_j \cdot K,$	$K \in \mathbb{Z} \setminus \{0\}$
div1:	$\mathrm{div1}(v_{\mathrm{neu}}, v_j, K)$	$\Leftrightarrow$	$v_{\mathrm{neu}} = v_j / K,$	$K \in \mathbb{Z} \setminus \{0\}$
mul2:	$\mathrm{mul2}(v_{\mathrm{neu}}, v_j, v_k)$	$\Leftrightarrow$	$v_{\mathrm{neu}} = v_j \cdot v_k$	
add2:	$\mathrm{add2}(v_{\mathrm{neu}}, v_j, v_k)$	$\Leftrightarrow$	$v_{\mathrm{neu}} = v_j + v_k$	
sub2:	$\mathrm{sub2}(v_{\mathrm{neu}}, v_j, v_k)$	$\Leftrightarrow$	$v_{\mathrm{neu}} = v_j - v_k$	

Hier bezeichnen v_j und v_k „alte" Variablen, die weiter links in der Klausel bereits aufgetreten sind, v_{neu} ist dagegen eine neue Variable, die durch das jeweilige Atom deterministisch festgelegt wird. Im folgenden werden die Atome dieser Typen auch als *abgeleitete Attribute* bezeichnet. Die Atome β_i $(1 \leq i \leq p)$ haben dagegen die Form

const:	$\mathrm{const}(v_j, d_1, d_2)$	$\Leftrightarrow$	$d_1 \leq v_j \leq d_2, \quad d_1, d_2 \in \mathbb{Z}$

mit entsprechender Bezeichnungskonvention. Sie legen fest, welche Attributvektoren von der Klausel c_i akzeptiert bzw. zurückgewiesen werden. Die *const*-Atome werden im folgenden *Bedingungen* oder *Prädikate* genannt.

Nach diesem Muster können schrittweise aus den Attributen arithmetische Terme gebildet werden, bis das Ergebnis eines Terms immer innerhalb eines festen Intervalls liegen muß. Atome der Form *add1* oder *sub1* (Addition/Subtraktion einer Konstanten) sind überflüssig, weil beliebige additive Konstanten in der Bedingung von *const* vorkommen können.

Für das schon mehrfach bemühte Beispiel $a(17), b(17, 42), c(59)$ könnte eine erlernte Klausel z. B. folgendermaßen aussehen, wobei abgeleitete Attribute und Bedingungen zur besseren Übersicht im Zusammenhang gruppiert sind:

$$\begin{aligned} \mathrm{ok}(v_1, v_2, v_3, v_4) \vdash\ & \mathrm{sub2}(v_5, v_1, v_2), \\ & \mathrm{const}(v_5, 0, 0), \\ & \mathrm{add2}(v_6, v_2, v_3), \\ & \mathrm{sub2}(v_7, v_6, v_4), \\ & \mathrm{const}(v_7, 0, 0). \end{aligned}$$

Die semantische Forderung an die von *LARGE* ausgegebene Hypothese lautet

$$H \models E,$$

d. h. alle beobachteten Wertekombinationen der Trainingsmenge müssen von der Hypothese akzeptiert werden.

Jede Klausel beschreibt eine Klasse von Fällen in der Trainingsmenge. Da die Hypothese H mehrere Klauseln umfassen kann, sind Disjunktionen unterschiedlicher Regeln darstellbar. Es wird *angestrebt*, daß jedes Lehrbeispiel $e \in E$ *genau eine* Klausel der Hypothese erfüllt, weil die Klauseln idealerweise disjunkte Klassen von Situationen beschreiben sollen. Wenn sich zwei Klassen überschneiden, liegt der Verdacht nahe, daß sie

1. noch nicht hinreichend speziell beschrieben sind oder
2. nicht wirklich unterschiedliche Fälle z. B. im Rahmen der Protokollsemantik definieren.

Allerdings können solche Überlappungen nicht formal ausgeschlossen werden, so daß auch Hypothesen mit mehrdeutiger Klassifizierung von Lehrbeispielen prinzipiell zulässig sind.

4.2.2 Grobalgorithmus

Die vom *LARGE*-Algorithmus durchgeführte Suche im Hypothesenraum muß gleichzeitig mehrere Teilaufgaben lösen, für die jeweils unterschiedliche Strategien erforderlich sind:

1. Mit Hilfe der Atome *mul1*, *div1*, *mul2*, *add2* und *sub2* werden schrittweise abgeleitete Attribute erzeugt.
2. Mit Hilfe des Atoms *const* werden Bedingungen über die Attribute der Trainingsmenge oder die abgeleiteten Attribute erzeugt, mit deren Hilfe eine Teilmenge der Trainingsmenge E als Treffermenge einer neuen Klausel ausgewählt wird.
3. Aus den so erzeugten Klauseln werden Klauselkombinationen ausgewählt, die die gesamte Trainingsmenge akzeptieren und dabei möglichst keine bzw. möglichst wenig Überschneidungen zwischen den Treffermengen der enthaltenen Klauseln verursachen.

Im folgenden wird der gesamte *LARGE*-Algorithmus zunächst grob skizziert. Die genaue Funktionsweise in Bezug auf die unterschiedlichen Teilaufgaben ist Thema der anschließenden Abschnitte. Im Algorithmus und in weiter unten angegebenen Formeln treten einige Parameter mit Bezeichnern in Großbuchstaben auf, die in einem weiteren Abschnitt genauer erläutert werden.

```
Algorithmus LARGE(E)
  E            : Beispieldaten (v_1, ... , v_n) aus n Eingabeattributen

  C, bestC     : Klauselmenge
  nC           : Klausel

  C, bestC ← ∅

  Wiederhole

    -- neue Klausel
```

$nC \leftarrow \emptyset$
Wähle $E' \subset E$ durch Ausschluß der Treffer einer Klausel $c \in C$
Wiederhole $N_CONDS$mal

-- neues Prädikat
Für alle Eingabeattribute v_k
Wiederhole NEW_ATTSmal

-- neues Attribut
Sortiere die Attribute v_i nach linearer Korrelation $\text{corr}(E'_i, E'_k)$
Suche v_i, v_j mit betragsmäßig größter Korrelation
$f \leftarrow$ linearer Regressionskoeffizient $\text{coeff}(E'_i, E'_k)$
Fallunterscheidung nach f

$+1$	$\Rightarrow$	$v_1^{new} \leftarrow v_i - v_k$
-1	$\Rightarrow$	$v_1^{new} \leftarrow v_i + v_k$
sonst	$\Rightarrow$	$v_1^{new} \leftarrow f \cdot v_i$

Fallunterscheidung nach $\text{coeff}(E'_i, E'_k) \cdot \text{coeff}(E'_j, E'_k)$

> 0	$\Rightarrow$	$v_2^{new} \leftarrow v_i + v_j$
sonst	$\Rightarrow$	$v_2^{new} \leftarrow v_i - v_j$

$v_3^{new} \leftarrow v_i \cdot v_j$
Wähle v_l^{new} nach Korrelation als neues Attribut

Für alle Attribute v_i
$nC_{i,1}^{new} \leftarrow nC \cup \min(E'_i) \leq v_i \leq \max(E'_i)$
$nC_{i,2}^{new} \leftarrow nC \cup v_i = x$ mit x häufigster Wert in E'_i
$nC_{i,3}^{new} \leftarrow nC \cup v_i \in \text{Cluster}(E'_i)$
$nC \leftarrow nC_{j,k}^{new}$ mit maximaler Klauselqualität
$E' \leftarrow \text{hits}(E', nC)$

$C \leftarrow C \cup nC$
-- Klauselauswahl
Suche beste Klauseln $C' \subseteq C$ mit $C' \models E$
Wenn C' gefunden und besser als $bestC$
$bestC \leftarrow C'$

bis $bestC \neq \emptyset$ und seit $TERM\_STEPS$ Schritten keine Verbesserung

Ausgabe: Klauselmenge $bestC$

Ende von *LARGE*

Der grundsätzliche Ablauf des Algorithmus ist der folgende: Es wird eine Liste C von Klauseln verwaltet. Pro Durchlauf der Hauptschleife wird jeweils eine neue Klausel nC schrittweise konstruiert. Dies erfolgt unter ausschließlicher Betrachtung einer Teilmenge E' der Trainingsmenge. Es werden abwechselnd neue Attribute und neue Bedingungen konstruiert und zu nC zugefügt. Alle Zwischenergebnisse für nC werden anhand einer globalen Qualitätsfunktion auf Klauseln verglichen, das beste Zwischenergebnis zu C zugefügt. Schließlich wird geprüft, ob sich ein neues Optimum für die Auswahl einer Klauselteilmenge $bestC$ aus

C ergibt, die alle Trainingsbeispiele akzeptiert.

4.2.3 Erzeugung der Attribute

Bei der Erzeugung der abgeleiteten Attribute besteht das Ziel darin, Terme zu konstruieren, die später einen konstanten bzw. wenig streuenden Wert zur Verwendung in einer Bedingung liefern. Als Kriterium zur Steuerung der Termbildung wurde der *lineare Korrelationskoeffizient* $r_{i,j}$ zwischen Attributen v_i, v_j gewählt:

$$r_{i,j} = \frac{\overline{(v_i - \overline{v_i})(v_j - \overline{v_j})}}{\sqrt{\overline{(v_i - \overline{v_i})^2} \cdot \overline{(v_j - \overline{v_j})^2}}} \tag{4.3}$$

Die Idee dahinter ist, daß sich im Falle einer Abhängigkeit wie z. B. $v_1 = v_2 + v_3$ ein vom Betrag her auffällig erhöhter linearer Korrelationskoeffizient zwischen v_1 und v_2 sowie zwischen v_1 und v_3 ergibt. Diese Beobachtung stimuliert daher die Neubildung eines Attributs $v_2 + v_3$ bei gleichem Vorzeichen der beiden Korrelationskoeffizienten bzw. $v_2 - v_3$ bei abweichenden.

Die Attributerzeugung wählt nacheinander alle v_k als potentielle Zielattribute des neuen Terms. Die restlichen Attribute v_i werden nach dem Betrag ihrer Korrelation mit v_k, $r_{i,k}$, absteigend sortiert. Aus den obersten beiden Elementen v_i, v_j, die dafür in Frage kommen, werden nach folgendem Schema neue Attribute konstruiert:

- Für Terme, die anhand des Zielattributs aus *einer* weiteren Variable gebildet werden, wird zunächst der lineare Regressionskoeffizient $c_{k,i}$ berechnet:

$$c_{k,i} = \frac{\overline{(v_i - \overline{v_i})(v_k - \overline{v_k})}}{\overline{(v_i - \overline{v_i})^2}} \tag{4.4}$$

  Sofern $c_{k,i}$ ganzzahlig gerundet einen Betrag von 1 hat, versucht der Algorithmus, mit einem der neuen zweistelligen Attribute $v_k - v_i$ bzw. $v_k + v_i$ je nach Vorzeichen von $c_{k,i}$ einen potentiell *konstanten* Term abzuleiten. Anderenfalls wird das einstellige Attribut $v_i \cdot c_{k,i}$ als potentiell zu v_k *ähnlicher* Term erzeugt. In beiden Fällen wird die Qualität des neuen Terms mit der Korrelation $r_{i,k}$ bewertet.

- Aus den beiden Variablen v_i und v_j werden die beiden neuen Attribute $v_i + v_j$ oder $-$ je nach Vorzeichen der Regressionskoeffizienten $-$ $v_i - v_j$ sowie $v_i \cdot v_j$ gebildet und jeweils mit der Korrelation zwischen dem neuen Attribut und v_k bewertet.

Das höchstbewertete der so gewonnenen Kandidatenattribute wird als neues Attributatom der neuen Klausel hinzugefügt. Dies alles geschieht mehrmals pro Zielattribut v_k, damit sich auch komplexere Terme bilden können.

4.2.4 Erzeugung der Bedingungen

Zur Erzeugung einer neuen Bedingung, d. h. eines Atoms des Typs *const*, werden für alle vorhandenen Attribute v_i folgende drei Bedingungen gebildet und versuchsweise der neuen Klausel nC hinzugefügt:

1. $\min\{E_i'\} \leq v_i \leq \max\{E_i'\}$, eine beschreibende Bedingung ohne diskriminierende Funktion.
2. $h_i \leq v_i \leq h_i$, wobei h_i der häufigste Wert in E_i' ist. Dies ist der Sonderfall einer *Einzelwertbedingung*.
3. $d_1 \leq v_i \leq d_2$ mit $(d_1, d_2) = \text{Cluster}(E_i')$, die Einschränkung auf ein Teilintervall, das mit Hilfe der heuristischen Funktion *Cluster* zum Auffinden auffälliger Häufungen bestimmt wird.

Anschließend wird diejenige neue Bedingung Teil der neuen Klausel, die die globale Qualität der neuen Klausel maximiert. Dies alles geschieht mehrmals pro neuer Klausel.

Die folgenden beiden Unterabschnitte erklären die Cluster-Suche und die Klausel-Qualitätsfunktion näher.

Heuristische Cluster-Suche

Bei der Auswahl eines Teilintervalls von Werten innerhalb aller vorkommender Werte für ein Attribut kann der *LARGE*-Algorithmus ja nicht auf Klassenkennzeichen zurückgreifen, wie es bei Verfahren nach dem Paradigma des überwachten Lernens möglich wäre. Das einzige Kriterium für *LARGE* ist das Auftreten auffälliger Häufungen von Werten. Eine solche auffällige Häufung ist rein anschaulich gegeben durch ein Intervall von Werten eines Attributs, *innerhalb* dessen relativ viele Beispiele liegen, *außerhalb* dessen Grenzen die Beispieldichte jedoch stark abnimmt.

Die heuristische Cluster-Suche erfolgt auf der Menge der in E_i' vorkommenden Attributwerte. Die gesuchte auffälligste Häufung ist gegeben durch dasjenige Intervall $[d_1; d_2]$, $d_1, d_2 \in E_i'$, das den *Häufungsquotienten* q maximiert:

$$q \stackrel{def}{=} \frac{\min\{d_1 - \max\{d_0 | d_0 \in E_i' \wedge d_0 < d_1\}, \min\{d_3 | d_3 \in E_i' \wedge d_3 > d_2\} - d_2\}}{\max\{d_R - d_L | d_1 \leq d_L < d_R \leq d_2 \wedge]d_L; d_R[\cap E_i' = \emptyset\}} \tag{4.5}$$

Der Zähler dieses Bruchs bezeichnet die Breite der kleineren Lücke unmittelbar „links“ bzw. „rechts“ neben dem fraglichen Intervall $[d_1; d_2]$, in der sich keinerlei Werte des Attributs befinden, während der Nenner die Breite der größten wertefreien Lücke im Innern des potentiellen Häufungsintervalls bezeichnet. Im Zähler soll der jeweilige Term im äußeren Minimum-Operator entfallen, sobald das $[d_1; d_2]$-Intervall am Rand des Wertebereichs liegt und daher keine gültiges d_0 bzw. d_3 existiert.

Zusammengenommen liefert q das Verhältnis aus der Grenzschärfe des Häufungsintervalls und seiner Kompaktheit, also ein gutes Maß für eine „auffällige“ Häufung. Figur A.39 zeigt ein Beispiel für die Berechnung von q anhand des Histogramms eines Attributes. Man beachte, daß periodische Histogramme wie in diesem Beispiel bei ganzzahligen Attributwerten leicht aufgrund von Multiplikationen auftreten können. Die Idee der Cluster-Suche besteht darin, die zu einer Klasse von Fällen gehörenden Werte innerhalb der gesamten Wertemenge E_i' auszumachen.

Entscheidend ist, daß das Auffinden des Häufungsintervalls mit maximalem q effizienter als durch erschöpfende Suche möglich ist, die aufgrund der zwei Intervallgrenzen quadratischen Aufwand in der Mächtigkeit der Wertemenge hätte. Dazu dient das folgende Theorem 6.

Theorem 6 *Sei* $]d_L; d_R[$ *die größte Lücke innerhalb des Wertebereichs* E_i',

$$]d_L; d_R[\cap E_i' = \emptyset \quad \wedge \quad d_R - d_L = \max\{b - a |\,]a; b[\cap E_i' = \emptyset\}.$$

$]d_L; d_R[$ *liegt außerhalb* eines *Intervalls* $[d_1; d_2]$ *mit* $d_1, d_2 \in E_i'$, *das* q *maximiert:*

$$d_L, d_R \notin]d_1; d_2[$$

Beweis: Seien q_L der Häufungsquotient zu $[min\{E_i'\}; d_L]$ und q^* der Häufungsquotient zu irgendeinem Intervall $[d_1; d_2]$, in dem $]d_L, d_R[$ enthalten ist. Es gilt

$$\begin{aligned} & q_L = \frac{d_R - d_L}{\Delta_I} \qquad \text{mit} \quad \Delta_I \leq d_R - d_L \\ \wedge \quad & q^* = \frac{\Delta_A}{d_R - d_L} \qquad \text{mit} \quad \Delta_A \leq d_R - d_L \\ \Rightarrow \quad & q_L = \frac{d_R - d_L}{\Delta_I} \geq \frac{d_R - d_L}{d_R - d_L} = 1 \geq \frac{\Delta_A}{d_R - d_L} = q^* \end{aligned}$$

Also ist $[min\{E_i'\}; d_L]$ ein Intervall, in dem $]d_L; d_R[$ nicht enthalten ist, und mindestens den gleichen Häufungsquotienten liefert. Das maximale q kann also immer mit einem Intervall erreicht werden, das $]d_L; d_R[$ nicht enthält. □

Dieses Theorem nutzt die Prozedur *Cluster* aus, indem sie den Wertebereich in die zwei Teile links und rechts der größten Lücke aufteilt, q für diese Teilintervalle berechnet und sich rekursiv für jeden der beiden Teile erneut aufruft:

Prozedur Cluster(a, b)

Wenn $a = b$ **Dann fertig**
Suche größte Lücke $]L; R[$ innerhalb $[a; b]$
Berechne $q(a, b)$ anhand $]L; R[$ und Außenränder
Cluster(a, L); Cluster(R, b)

Ende von Cluster

Das Vermerken der Intervallgrenzen für den maximalen Häufungsquotienten wurde hier weggelassen. Der mittlere Aufwand für *Cluster* sinkt durch diese Implementierung auf $O(n \log n)$ in der Mächtigkeit der Wertemenge.

Klausel-Qualitätsfunktion

Für die Bewertung der Qualität einer Klausel wurde eine Klausel-Qualitätsfunktion $Q : K \rightarrow \mathbb{N}$ definiert, wobei K die Menge der möglichen Klauseln bezeichnet. Q wird zu drei verschiedenen Zwecken innerhalb *LARGE* verwendet:

1. Um bei der Konstruktion neuer Bedingungen die für die neue Klausel jeweils beste zu bestimmen (vgl. Abschnitt 4.2.4).

2. Um von den p Zwischenergebnissen

$$\begin{aligned} \mathrm{ok}(v_1,\ldots,v_n) &\vdash \ldots,\beta_1. \\ \mathrm{ok}(v_1,\ldots,v_n) &\vdash \ldots,\beta_1,\beta_2. \\ &\vdots \\ \mathrm{ok}(v_1,\ldots,v_n) &\vdash \ldots,\beta_1,\beta_2,\ldots,\beta_p. \end{aligned}$$

das beste als tatsächlich in der Klauselliste C zu behaltende neue Klausel auszuwählen (vgl. Abschnitt 4.2.2).

3. Um bei der Klauselauswahl für die auszugebende Hypothese (siehe nächster Abschnitt) die besten Klauseln aus C mit geeigneter Klassifizierung der Trainingsbeispiele zu wählen.

Aufgrund dieser Anwendungen von Q ist es entscheidend, daß $Q(c)$ jede Klausel *einzeln* bewertet, ohne Rücksicht auf die Akzeptanzmengen der anderen Klauseln oder den Beitrag von c zu einer vollständigen und überlappungsfreien Akzeptanz der Trainingsmenge.

Sei c eine Klausel

$$c = \mathrm{ok}(v_1,\ldots,v_n) \vdash \alpha_1,\ldots,\alpha_q,\beta_1,\ldots,\beta_p.,$$

mit den *Bedingungen* β_i. Die Atome α_i, die die *abgeleiteten Attribute* bilden, spielen für die Berechnung der Klauselqualität keine direkte Rolle, sondern definieren nur zusätzliche Attribute, auf die in den β_i zugegriffen wird. Außerdem bezeichne $T_c \subseteq E$ die Treffermenge der Klausel,

$$T_c \stackrel{def}{=} \{e \in E | c \models e\},$$

und $T_i \subseteq E$ die Treffermenge der Bedingung β_i,

$$T_i \stackrel{def}{=} \{e \in E | \beta_i \models e\} \quad \text{mit} \quad 1 \leq i \leq p.$$

Die Klauselqualität $Q(c)$ ist definiert als

$$Q(c) \stackrel{def}{=} 100 \cdot attr \cdot rej + compl. \tag{4.6}$$

Die einzelnen Komponenten berechnen sich wie folgt:

- *attr* (für *attraction*, Auffälligkeit) ist ein Maß für die mittlere Auffälligkeit der von den einzelnen Bedingungen ausgewählten Werte des jeweiligen Attributs innerhalb der vorkommenden Werte. Seine Definition ist:

$$attr \stackrel{def}{=} \frac{1}{p}\sum_{i=1}^{p} \min\{99; 99 \cdot \mathrm{att}(\beta_i)\}.$$

  Die Bewertungsfunktion att für die einzelnen Bedingungen unterscheidet zwischen den folgenden drei Fällen:

1. $\beta_i = \text{const}(v_s, d, d)$ wählt einen *Einzelwert* d aus: Seien $T^* \stackrel{def}{=} T_1 \cap \ldots \cap T_{i-1}$ die Treffermenge der Klauselbedingungen vor β_i, $N(w)$ die absolute Häufigkeit des Wertes w im Attribut v_s in T^*, also

$$N(w) \stackrel{def}{=} \left|\{j \mid (T^*)_s^j = w\}\right|,$$

und N^* die Anzahl der *übrigen* Einzelwerte, deren Auftretenshäufigkeit in T^* den Bruchteil *SINGLE_EXCEL* der Häufigkeit von d erreicht oder überschreitet:

$$N^* \stackrel{def}{=} |\{w \in \mathbb{Z} | N(w) \geq SINGLE\_EXCEL \cdot N(d)\}| - 1$$

Dann ist

$$\text{att}(\beta_i) \stackrel{def}{=} SINGLE\_F \cdot \frac{\max\{0; 1 - COMPARE\_F \cdot N^*\}}{1 + \log_{10}(|T^*| / N(d))}.$$

Im Zähler des Bruchs steht ein Maß für die Anzahl sonstiger Einzelwerte, die „fast so oft" wie d auftreten, und im Nenner für den Anteil der von β_i ausgewählten Beispiele an der noch übrigen Treffermenge. Im Idealfall wird $N^* = 0$ und $N(d) = |T^*|$, der Bruch also zu 1.

2. Sonst, wenn $\beta_i = \text{const}(v_s, d_1, d_2)$ ein Intervall auswählt und dabei Beispiele ausschließt, also $T_1 \cap \ldots \cap T_i \neq T^*$: Bezeichne N^{**} die Anzahl der *übrigen* Intervalle, deren logarithmisch bewerteter Häufungsquotient den Bruchteil *CLUSTER_EXCEL* der Bewertung von $[d_1; d_2]$ erreicht oder überschreitet:

$$N^{**} \stackrel{def}{=} |\{(v, w) \in T_s^{*2} | \log_{10} q_{[v;w]} \geq CLUSTER\_EXCEL \cdot \log_{10} q_{[d_1;d_2]}\}| - 1$$

Damit berechnet sich

$$\begin{aligned}\text{att}(\beta_i) \stackrel{def}{=} CLUSTER\_F \quad &\cdot \max\{0; \log_{10} q_{[d_1;d_2]}\} \\ &\cdot \max\{0; 1 - COMPARE\_F \cdot N^{**}\}\end{aligned}$$

mit q jeweils als dem Häufungsquotienten gemäß (4.5) auf der Beispielmenge T^*. Der zweite *max*-Term begrenzt auch hier die Bewertung für Bedingungen, die eine von vielen „fast gleichguten" Auswahlen treffen. Die Berechnung von N^{**} erfolgt tatsächlich während der rekursiven Ausführung der *Cluster*-Prozedur und verursacht daher nur $n \log n$ Aufwand. Das ist zulässig, weil alle von der Berechnung ausgeschlossenen Vergleichsintervalle einen Häufungsquotienten von höchstens 1 haben (siehe Beweis von Theorem 6) und daher nur für Intervalle $[d_1; d_2]$ eine Rolle spielen würden, die ohnehin mit 0 bewertet werden.

3. Sonst, wenn $T_i \supseteq T^*$, d. h. $\beta_i = \text{const}(v_s, d_1, d_2)$ das volle Werteintervall seines Attributs akzeptiert, gilt:

$$\text{att}(\beta_i) \stackrel{def}{=} FORCE\_F \cdot (FULL\_F0 + FULL\_F / \log_{10}(d_2 - d_1 + 1))$$

Es werden also schmalere Intervalle gegenüber breiteren belohnt, wobei der Parameter *FULL_F0* eine breitenunabhängige Grundbewertung modelliert. *FORCE_F*

ist ein dynamischer Parameter, der verwendet wird, um die Termination von *LARGE* sicherzustellen: Wenn die Anzahl der gebildeten Klauseln ihr Maximum *CLAUSES_MAX* erreicht hat, ohne daß alle Trainingsbeispiele akzeptiert sind, wird *FORCE_F* von initial 1 schrittweise erhöht, um Klauseln zu bevorzugen, die wenig Beispiele ausschließen.

- *rej* (für *rejections*, Ablehnungen) gibt an, wie viele Bedingungen der Klausel im Mittel pro abgelehntem Trainingsbeispiel unerfüllt sind:

$$rej \stackrel{def}{=} \begin{cases} \min\left\{99; \left\lfloor \frac{10}{|E \setminus T_c|} \sum_{i=1}^{p} |E \setminus T_i| \right\rfloor\right\} & \text{wenn } T_c \neq E \\ \min\{99; \lfloor 10 \cdot FORCE\_F \cdot FULL\_RED \cdot p \rfloor\} & \text{wenn } T_c = E \end{cases}$$

  Bei Klauseln, die nur eine Bedingung haben und Beispiele ausschließen, ergibt sich immer der Wert 10. *rej* nimmt zu, wenn die von der Klausel akzeptierte Beispielklasse durch *mehrere verschiedene Bedingungen*, also redundant, definiert wird. Solche Redundanz in den Klassifikationregeln wird als Argument dafür verstanden, daß die gefundenen Klassengrenzen mit Auffälligkeiten innerhalb der Trainingsmenge übereinstimmen. Figur A.40 stellt diesen Aspekt grafisch dar: Im Figurteil a lassen sich die im eingezeichneten Rechteck liegenden Punkte durch jede der beiden Bedingungen β_1 und β_2 bereits weitgehend klassifizieren, die Klassifikation durch β_1 *und* β_2 ist teilweise redundant. Im Teil b ist die Beispielhäufung innerhalb der von c definierten Klasse weniger auffällig, es sind beide Bedingungen erforderlich. Die Werte $rej = 17$ bzw. $rej = 13$ geben an, daß die *nicht* zur von c definierten Klasse gehörenden Beispiele im Mittel durch $1,7$ bzw. $1,3$ *const*-Atome in c abgelehnt werden.

  Für Klauseln, die nicht diskriminieren, sondern die gesamte Trainingsmenge beschreiben, kann kein Maß für die abgelehnten Beispiele errechnet werden. Dieser Fall $T_c = E$ sollte idealerweise immer dann eintreten, wenn alle Beispiele tatsächlich derselben Klasse zugeordnet werden können – die Klassenzahl ist dem *LARGE*-Algorithmus ja nicht vorgegeben. Diese Klauseln werden mit einem Vergleichswert bewertet, der einerseits proportional zur Klauselläge ist, weil ja auch die erzielbare Redundanzbewertung von diskriminierenden Klauseln mit deren Länge zunimmt. Der dynamische Faktor *FORCE_F* dient wiederum zur Beschleunigung der Termination, falls *LARGE* am Anfang zu kleine Klassen findet. Mit *FULL_RED* kann die Gewichtung von beschreibenden gegenüber diskriminierenden Klauseln statisch festgelegt werden.

- *compl* (für *complexity*, Komplexität der Klausel) soll einfachere Klauseln im Sinne des MDL-Paradigmas (vgl. Abschnitt 4.1.2) belohnen. Diese Komponente berechnet sich rekursiv über den Aufbau der abgeleiteten Attribute und gibt die mittlere Anzahl der Operatoren pro Bedingungsattribut an. Wegen der Interpretation von Q als Qualitätsfunktion muß *compl* mit steigender Komplexität abnehmen, weshalb der Wert so umgerechnet wird, daß $compl = 99$ für keinen Operator steht, d. h. nur Eingabeattribute in den Bedingungen, und $compl = 0$ für durchschnittlich $9,9$ und mehr Operatoren pro

Bedingungsterm, mit linearer Interpolation dazwischen:

$$compl \stackrel{def}{=} \frac{1}{p}\sum_{i=1}^{p}\max\{0; 99 - 10 \cdot \mathrm{cmp}(v_{\beta_i})\}$$

mit

$$\mathrm{cmp}(v_j) \stackrel{def}{=} \begin{cases} \mathrm{cmp}(v_s) + 1 + \mathrm{cmp}(v_t) & \text{falls } \mathrm{mul2}(v_j, v_s, v_t) \in c \\ \mathrm{cmp}(v_s) + 1 + \mathrm{cmp}(v_t) & \text{falls } \mathrm{add2}(v_j, v_s, v_t) \in c \\ \mathrm{cmp}(v_s) + 1 + \mathrm{cmp}(v_t) & \text{falls } \mathrm{sub2}(v_j, v_s, v_t) \in c \\ \mathrm{cmp}(v_s) + 1 & \text{falls } \mathrm{mul1}(v_j, v_s, k) \in c \\ \mathrm{cmp}(v_s) + 1 & \text{falls } \mathrm{div1}(v_j, v_s, k) \in c \\ 0 & \text{falls } 1 \leq j \leq n \end{cases}$$

Dabei bezeichnet v_{β_i} das in der Bedingung β_i getestete Attribut.

Die Einschränkung des Wertebereichs der Komponenten *attr*, *rej* und *compl* auf $[0; \ldots; 99]$ und der Gewichtungsfaktor 100 in (4.6) sorgen dafür, daß die Komponenten *attr* und *rej* bei Vergleichen absolute Priorität vor der Komponente *compl* erhalten. *attr* und *rej* sind dagegen durch die Multiplikation gleichberechtigt und arithmetisch UND-verknüpft. Werte der Qualitätsfunktion tragen in den Einer- und Zehnerstellen die Bewertung der Termkomplexitäten, in den Hunderter- bis Hunderttausenderstellen das Produkt aus den Bewertungen der Cluster-Auffälligkeit und der Klassifikationsredundanz.

4.2.5 Auswahl der Klauseln

Bisher wurde beschrieben, wie Atome und Klauseln konstruiert und damit die Klauselmenge C im Algorithmus schrittweise aufgebaut werden. Der letzte Schritt ist nun, aus der aktuellen Klauselmenge C eine Auswahl von Klauseln zur Hypothese $h \subseteq C$ zusammenzustellen, die alle Beispiele der Trainingsmenge E akzeptiert (vgl. Abschnitt 4.2.1).

Für die Klauselauswahl ist die Klauselliste $C = c_1, \ldots, c_k$ nach der Klauselqualität $Q(c)$ (4.6) absteigend sortiert. Es wird eine Teilmenge $h = c_1^h, \ldots, c_l^h$ von Klauseln gesucht, die möglichst wenige Treffer t_h auf der Trainingsmenge aufweist, aber alle Lehrbeispiele akzeptiert:

$$t_h \stackrel{def}{=} \sum_{i=1}^{l} |\{e \in E | c_i^h \models e\}$$

$$t_h \geq |E| \quad \wedge \quad t_h \text{ minimal!}$$

Zum Vergleich solcher Klauselauswahlen $h \subseteq C$ ist eine Qualitätfunktion Q^* für Hypothesen definiert:

$$Q^*(h) \stackrel{def}{=} \begin{cases} \sum_{\substack{i=1 \\ c_i \in h}}^{k} 2^{CLAUSES\_MAX - i} & \text{wenn } t_h = |E| \\ \frac{1}{t_h - |E|} & \text{wenn } t_h > |E| \end{cases} \tag{4.7}$$

Durch diese Definition sind überlappungsfreie Hypothesen (erster Fall) immer besser als solche mit mehrdeutiger Klassenzuordnung (zweiter Fall). Bei überlappungsfreien Hypothesen werden Klauseln aus dem vorderen Teil von C belohnt, also die mit der höchsten Klauselqualität, ansonsten die mit dem geringeren Überlappungsgrad $t_h - |E|$.

Der Algorithmus zur Auswahl von Klauselteilmengen trägt zunächst all jene Klauseln in die Hypothese ein, die irgendein Beispiel als einzige klassifizieren und daher mit Sicherheit in der Hypothese enthalten sein müssen. Im ersten Durchlauf versucht er dann, eine optimale Überlappungfreie Hypothese zu konstruieren. Dazu werden von vorn nach hinten alle Klauseln aus C, die nicht zu Mehrfachtreffern in der Hypothese führen, hinzugenommen. Wenn die Suche das Ende der Liste erreicht und die aktuelle Hypothese h noch nicht alle Beispiele abdeckt, erfolgt Backtracking nach dem folgenden Schema: Die Suche von vorn nach hinten wird wiederaufgenommen an der dem Listenende nächsten Position i mit $c_i \in h, c_{i+1} \notin h$, nachdem alle Klauseln vom Listenende bis einschließlich c_i wieder aus der Hypothese gestrichen worden sind. Sobald eine vollständige überlappungsfreie Hypothese gebildet wurde, stellt sie die gesuchte optimale Auswahl gemäß (4.7) dar, was die geschilderte Suchstrategie und die Vorsortierung der Klauseln garantieren. Wenn keine solche Hypothese gefunden werden konnte, erfolgt ein zweiter Durchlauf, um die beste Auswahl *mit* Überlappung zu ermitteln. Die Suchstrategie ist zum ersten Durchlauf weitgehend analog, allerdings kann die Suche nicht mit dem ersten Fund einer vollständigen Beispielabdeckung abgebrochen werden, da ja gemäß (4.7) die Gesamttrefferzahl t_h zu minimieren ist.

Um den Aufwand bei der Suche nach geeigneten Klauselteilmengen zu begrenzen, der normalerweise bei $O(|\mathbb{P}C|) = O(2^k)$ läge, wird jeweils nach *SELECT_MAX* Schleifendurchläufen abgebrochen.

Die so konstruierte Hypothese wird zur aktuellen Ausgabehypothese, falls ihr $Q^*(h)$ das der letzten Ausgabehypothese übersteigt.

4.2.6 Suchstrategie

Die bisherigen Erklärungen zur Klauselkonstruktion würden nicht ausschließen, daß *LARGE* Zyklen im Suchraum durchläuft oder immer wieder die gleichen Attribute oder mathematisch äquivalente Bedingungen bildet. Daher gibt es eine Reihe von Regeln und Maßnahmen, die die Suche so steuern, daß die genannten unerwünschten Effekte ausgeschlossen sind.

Vorauswahl von Beispielen

Die erste dieser Maßnahmen ist die Vorauswahl der Beispielteilmenge E' im Algorithmus (siehe Abschnitt 4.2.2) – dort als „wähle $E' \subset E$ durch Ausschluß der Treffer einer Klausel $c \in C$" bezeichnet. Hierzu verwendet *LARGE* einen Klauselindex *cls_excl*, der zyklisch alle Klauseln durchläuft. Alle Treffer der aktuell indizierten Klausel gelangen nicht in die Menge E', die den Ausgangspunkt zur Bildung der nächsten neuen Klausel bildet. Pro Durchlauf des Index *cls_excl* wird genau einmal die vollständige Beispielmenge E herangezogen:

$$E' = \begin{cases} E \setminus T_{c_{cls\_excl}} & \text{wenn } 1 \leq cls\_excl \leq |C| \\ E & \text{wenn } cls\_excl = 0 \end{cases}$$

Dadurch werden Beispiele, die gerade durch eine neue Klausel zu einer möglichen Klasse zusammengefaßt worden sind, bei einer der nächsten Klauselbildungen gewissermaßen ausgeblendet. Somit können auch Klassengrenzen in solchen Dimensionen des Attributraums gefunden werden, in denen sich bei Betrachtung der gesamten Trainingsmenge keine zur Klassendiskrimination geeigneten Häufungen zeigen. Figur A.41 stellt einen solchen Fall beispielhaft dar: Im Teil a kann nur die Menge A diskriminiert werden, weil entlang der Achse v_j kein Dichteunterschied erkennbar ist. Erst nach Ausblenden von A erfolgt die Diskrimination der Menge B (Teil b).

Tabu-Suche

Die zweite Maßnahme bei der Suchstrategie betrifft die Erzeugung von Bedingungen (siehe Abschnitt 4.2.4): Hier verwendet *LARGE* das Verfahren der Tabu-Suche (*tabu search*), um nicht wiederholt die gleichen Diskriminationen zu liefern. Zu diesem Zweck wird eine Tabuliste *set_tabu* von Beispielklassen verwaltet. Diese Liste ist aus Effizienzgründen mit Hilfe einer Hash-Funktion auf E-Teilmengen implementiert, was vom Prinzip her jedoch keinen Unterschied macht:

- Bei der schrittweisen Erweiterung der neuen Klausel nC um neue Bedingungen werden alle Bedingungen verworfen, die zu einer von der Klausel akzeptierten Beispielmenge führen, die in *set_tabu* vermerkt ist.
- Immer wenn C bereits *MAX_CLAUSES* umfaßt und die letzte neuerzeugte Klausel nC von ihrer Klauselqualität $Q(nC)$ her schlechter ist als die beste Klausel in C, wird die von nC diskriminierte Klasse in die Tabuliste eingetragen.
- Wenn *set_tabu* mehr als *TABUS_MAX* Einträge umfaßt, werden der oder die ältesten davon gelöscht, bis die Maximallänge wieder eingehalten wird.

Regeln für die Atome

Da bei der Bildung abgeleiteter Attribute grundsätzlich beliebige arithmetische Terme entstehen können, besteht die Gefahr, daß *LARGE* redundante Attribute und äquivalente Klassifikationsregeln erzeugt. Ein Beispiel dafür wäre das folgende, mit den abgeleiteten Attributen als Terme ausgeschrieben:

$$\begin{aligned} \mathrm{ok}(v_1, v_2, v_3) \quad &\Leftrightarrow \quad v_1 - v_2 = v_3 \\ &\wedge \quad v_1 + (-1 \cdot v_3) = v_2 \end{aligned}$$

Durch die Belohnung redundanter Diskrimination in der Klausel-Qualitätsfunktion Q (Komponente *rej* in 4.6, siehe Abschnitt 4.2.4) würden derart unsinnige Suchergebnisse sogar bevorzugt werden. Die hier vorliegende Redundanz ist aber wertlos, weil sie von zwei identischen Bedingungen herrührt. Die von Q sinnvollerweise zu belohnende Redundanz betrifft jedoch übereinstimmende Diskriminationseffekte *verschiedener* Eingabeattribute und ist damit eine Eigenschaft der Trainingsmenge, nicht der Klassifikationsregeln.

Die Möglichkeit, Äquivalenzen dieser Art durch Integration eines Theorembeweisers in *LARGE* aufzuspüren, muß aus Komplexitätsgründen verworfen werden. Stattdessen gilt der folgende Satz von Regeln bei der Konstruktion von Klauseln:

1. Als Zielattribute bei der Bildung von Attribut-Atomen mit Hilfe der Korrelationskoeffizienten sind nur die Eingabeattribute $v_1, \ldots, v_n$ zugelassen. Dadurch wird die Anzahl der Ziele für die Termbildung beschränkt, ohne an Ausdrucksfähigkeit zu verlieren.

2. Wenn ein Eingabeattribut v_k in die Berechnung eines abgeleiteten Attributs v_i eingeht, wird v_i nicht benutzt, um ein abgeleitetes Attribut mit Ziel v_k zu bilden. Denn diese Korrelation wäre ungerecht hoch und die resultierende Regel ohnehin verboten wegen Punkt 4.

3. Ein mit *mul1* gebildetes Attribut der Form $v_i \cdot k$ wird nie als Argument eines weiteren *mul1*-Atoms in Betracht gezogen, da dasselbe Ergebnis durch direkte Verwendung von v_i in einem *mul1*-Atom mit anderem Faktor k erreichbar ist.

4. Zweistellige Attribut-Atome der Typen *mul2*, *add2* und *sub2* mit Argumenten v_i, v_j werden nicht gebildet, wenn v_i und v_j aus ein oder mehreren gemeinsamen Eingabeattributen berechnet sind. Sonst könnten einzelne Eingabeattribute durch inverse Operationen aus dem Ergebnis faktisch eliminiert werden, wie etwa in $(v_1 + v_2) - v_1$.

5. Alle Bedingungen einer Klausel müssen sich auf *verschiedene* (nicht notwendig paarweise disjunkte!) Mengen von Eingabeattributen beziehen. Dies ist die entscheidende Regel zur Vermeidung äquivalenter Diskriminationsregeln innerhalb einer Klausel.

6. Je nach Anwendungsfall ist es nicht sinnvoll, Terme mit mehrfach verketteten Multiplikationen zu bilden. Dies kann erstens Überläufe des Zahlenbereichs in der Implementierung provozieren, zweitens bilden sich durch Multiplikationen immer ungleichmäßige Verteilungen der auftretenden Werte des Produktattributs, was die Suche nach systematischen Häufungen erschwert. Daher wird die Multiplikationstiefe der abgeleiteten Attribute auf *MULS_MAX* beschränkt. Als Multiplikationstiefe eines Attributatoms gilt hier das Maximum der Multiplikationstiefen seiner Argumente, genau dann um 1 erhöht, wenn es sich um ein *mul1*- oder *mul2*-Atom handelt.

Termination

LARGE terminiert, wenn eine alle Beispiele abdeckende Hypothese gefunden wurde und *TERM_STEPS* Klauselkonstruktionen lang keine Verbesserung der Hypothesenqualität gemäß Q^* erzielt werden konnte.

4.2.7 Parameter

Die folgende Tabelle zählt die Parameter auf, die den *LARGE*-Algorithmus steuern. Viele davon sind in den vorangegangenen Abschnitten bereits in Formeln oder Teilalgorithmen aufgetreten:

Parameter	Beschreibung	Vorgabe
CLUSTER_MIN	Minimale Anzahl von Datenpunkten in einem Cluster, damit auch minimale Klassengröße.	5
TERM_STEPS	Nach *TERM_STEPS* Hauptschleifendurchläufen ohne Verbesserung der Hypothese erfolgt die Termination.	6
SELECT_MAX	Maximale Anzahl Durchgänge bei der Suche nach einer Klauselauswahl für die Hypothese, nötig zur Aufwandsbegrenzung.	50
TABUS_MAX	Maximal *TABUS_MAX* Einträge bleiben in der Klassen-Tabuliste stehen, wenn eine neue Klausel gefunden wurde.	4
MULS_MAX	Maximale Zahl von verketteten Multiplikationen in einem abgeleiteten Attribut, bildlich die maximale Zahl von Multiplikationen in den Pfaden des Berechnungsbaums mit den Eingabeattributen als Blättern und dem Ergebnis als Wurzel.	1*
DIS_MAX	Maximale Zahl von *Diskriminationen* pro Klausel. Eine Diskrimination erfolgt durch die k-te Bedingung genau dann, wenn ihre Treffermenge mindestens ein Beispiel aus der Treffermenge aller bisherigen Bedingungen ausschließt, formal: $T_1 \cap \ldots \cap T_k \subset T_1 \cap \ldots \cap T_{k-1}$.	2
SINGLE_F	Gewichtung der Auffälligkeitsbewertung für Einzelwerte in Q.	1,5
SINGLE_EXCEL	$SINGLE\_EXCEL \cdot N(d)$ ist die Schwelle für die Häufigkeit eines anderen Attributwertes w in der Beispielmenge, ab der w in der Auffälligkeitsbewertung als „fast so häufig wie d“ gilt. Dies mindert die Auffälligkeit von d.	0,4
CLUSTER_F	Gewichtung der Auffälligkeitsbewertung für Intervall-Cluster in Q.	0,5
CLUSTER_EXCEL	$CLUSTER\_EXCEL \cdot \log_{10} q$ ist die Schwelle für den Zehnerlogarithmus des Häufungsquotienten eines anderen Intervalls $[d_1; d_2]$ in der Beispielmenge, ab der $[d_1; d_2]$ in der Auffälligkeitsbewertung als „fast so häufig wie das gewählte Intervall“ gilt. Dies mindert die Auffälligkeit des gewählten Intervalls.	0,5
COMPARE_F	Steilheit der Abwertung bei „ähnlich auffälligen“ konkurrierenden Einzelwerten oder Clustern. Ab $1/COMPARE\_F$ Konkurrenten ist die Auffälligkeit null.	0,2

Parameter	Beschreibung	Vorgabe
FULL_F0	Konstante Auffälligkeitsbewertung für Intervallbedingungen, die alle Beispiele ihrer Ausgangmenge erfassen, d. h. nicht diskriminieren.	0,1
FULL_F	Intervallbreitenabhängige Auffälligkeitsbewertung für Intervallbedingungen, die alle Beispiele ihrer Ausgangmenge erfassen, d. h. nicht diskriminieren.	0,2
FULL_RED	Faktor für die Redundanzbewertung nichtdiskriminierender Intervallbedingungen. Bei $1,0$ entspricht die Bewertung einer maximal redundanten diskriminierenden Klausel.	0,75
FORCE_STEP	Faktor für die schrittweise Erhöhung der Bewertung nichtdiskriminierender Klauseln, wenn die Klauselliste schon gefüllt ist, aber die aktuelle Hypothese noch nicht alle Beispiele akzeptiert.	1,5
N_CONDS	Maximale Anzahl Bedingungen pro Klausel.	$n+1$
ATTS_MAX	Maximale Anzahl von Attributen insgesamt.	$2n \cdot NEW\_ATTS$
CLAUSES_MAX	Maximale Anzahl Klauseln in der Klauselliste bzw. der Ausgabehypothese.	10*
NEW_ATTS	Anzahl Termneubildungen pro neuer Bedingung.	15*
ATT_IGNORE	Neugebildete Attribute werden erst *ATT_IGNORE* Attributbildungen später als Argumente herangezogen. Steuert die Tiefen- und Breitensuche-Anteile bei der Termbildung.	3*

Die mit * gekennzeichneten Parameter sind diejenigen, die auch für Änderungen durch den Anwender bei konkreten Lernaufgaben vorgesehen sind. n ist die Anzahl der Eingabeattribute $v_1, \ldots, v_n$.

4.2.8 Aufwandsabschätzung

In diesem Abschnitt erfolgt eine Abschätzung des Laufzeitaufwands im schlimmsten Fall, C_{LARGE}, für einen einzelnen Schritt der *LARGE*-Hauptschleife, also die Konstruktion einer neuen Klausel und die Klauselauswahl für die aktuelle Hypothese.

Bei der Aufwandsberechnung wird die Anzahl der Maschinenoperationen üblicher Prozessoren als Berechnungsgrundlage verwendet. Daher sind die Schleifenstrukturen des bis auf solche Elementaroperationen hinunter verfeinerten Algorithmus zu analysieren. Die bei dieser Analyse im ersten Schritt auftretenden freien Variablen sind in Tabelle 4.2 beschrieben.

Die Analyse der Schleifenstruktur liefert folgende Terme für die Berechnung von C_{LARGE}:

- Auswahl der Beispielteilmenge E' für die Klauselkonstruktion: Die Auswahl erfolgt in einer Schleife über die Trainingsmenge in Abhängigkeit von der aktuell auszuschließenden Beispielklasse. Unter Vernachlässigung multiplikativer Konstanten ergibt sich als Aufwandsterm:

$$C^1_{LARGE} = O(m) \tag{4.8}$$

- Berechnung der Korrelations- und Regressionskoeffizienten und Sortierung der Attribute: Die Korrelationsberechnung erfolgt für jedes Zielattribut einzeln, wofür nur Eingabeattribute verwendet werden.

  Der resultierende Aufwandsterm ist:

$$\begin{aligned} C^2_{LARGE} &= O(n \cdot ATTS\_MAX \cdot m + n \cdot ATTS\_MAX \log ATTS\_MAX) \\ &= O((n \cdot ATTS\_MAX) \cdot (m + \log ATTS\_MAX)) \end{aligned} \tag{4.9}$$

- Suche von Kandidaten für die Attributneubildung: Hier treten bis zu vier geschachtelte Schleifen auf, nämlich über die Zielattribute v_k, *NEW_ATTS* Neubildungen und das Paar v_i, v_j der Argumentattribute. Für v_i und v_j kommen jeweils die n Eingabeattribute und a_k der abgeleiteten Attribute in Frage, dies sind diejenigen abgeleiteten Attribute mit Zielattribut v_k. Die zugehörige Summe $(n + a_k)$ kann grob mit *ATTS_MAX* nach oben abgeschätzt werden.

  Der resultierende Aufwandsterm ist:

$$\begin{aligned} C^3_{LARGE} &= O\Big(\sum_{k=1}^{n} NEW\_ATTS \cdot (n + a_k)^2 \cdot n\Big) \\ &= O(NEW\_ATTS \cdot ATTS\_MAX^2 n^2) \end{aligned} \tag{4.10}$$

  Das n aus der innersten Schleife stammt vom Test auf gemeinsame Eingabeattribute in den Termen zu v_i und v_j. Man beachte, daß die Suchschleife für v_i, v_j nur solange durchlaufen wird, bis ein Argumentpaar gegen keine der Regeln aus Abschnitt 4.2.6 verstößt. Dieses Paar wird verwendet, weil es sich ja dank der Sortierung nach Korrelation zu v_k um das am besten passende Attributpaar handelt. In Fällen, wo die obersten beiden Attribute verwendbar sind, reduziert sich der Aufwand dieses Schritts daher auf $O(NEW\_ATTS \cdot n^2)$.

- Eintragen des neuen Attributs in die Attributtabelle: Dabei müssen alle Werte des abgeleiteten Attributs für sämtliche Lehrbeispiele berechnet werden. Außerdem ist die Korrelation des neuen Attributs mit dem aktuellen Zielattribut zu berechnen und in die Liste mit den Korrelationswerten einzusortieren. Es ergibt sich:

$$C^4_{LARGE} = O(n + ATTS\_MAX + m \log m) \tag{4.11}$$

  Der Logarithmus ist eine Folge davon, daß die Wertmenge des neuen Attributs – zwecks leichterer Auswertung z. B. des Wertehistogramms – sortiert abgelegt wird.

- Suche nach den neuen Bedingungen: Die Suchschleife läuft hier über alle vorhandenen maximal *ATTS_MAX* Attribute und wird *N_CONDS*-mal ausgeführt. Entscheidend für den Aufwand dieses Schritts ist die rekursive Cluster-Suche (vgl. Abschnitt 4.2.4), die sowohl bei der Auswahl der Cluster-Intervalle als auch zur Berechnung der Klauselqualitätsfunktion benötigt wird. Zusammen mit den Tests der Regeln für Attribute, insbesondere daß jede Bedingung einen anderen Satz von Eingabeattributen verwendet, errechnet sich der Aufwand zu:

$$C^5_{LARGE} = O(N\_CONDS \cdot ATTS\_MAX \cdot (N\_CONDS \cdot n + m \log m)) \quad (4.12)$$

- Auswahl einer Klauselmenge als Hypothese: Die Suche nach geeigneten Teilmengen ist auf *SELECT_MAX* Durchläufe begrenzt. Aufwandsentscheidend ist, daß beim Streichen und Wiedereinfügen von Klauseln aus der bzw. in die Hypothese die Klauseltrefferzahlen aller Lehrbeispiele aktualisiert werden müssen:

$$C^6_{LARGE} = O(SELECT\_MAX \cdot cm) \quad (4.13)$$

Zusammen ergibt sich für den Zeitaufwand eines *LARGE*-Suchschritts folgende Aufwandsklasse, wobei die aktuelle Klauselzahl c noch durch ihre obere Schranke *CLAUSES_MAX* abgeschätzt ist:

$$\begin{aligned} C_{LARGE} &= C^1_{LARGE} + C^2_{LARGE} + C^3_{LARGE} + C^4_{LARGE} + C^5_{LARGE} \\ &= O((n \cdot ATTS\_MAX) \cdot (m + \log ATTS\_MAX) \\ &\quad + NEW\_ATTS \cdot ATTS\_MAX^2 n^2 \\ &\quad + N\_CONDS \cdot ATTS\_MAX \cdot (N\_CONDS \cdot n + m \log m) \\ &\quad + SELECT\_MAX \cdot CLAUSES\_MAX \cdot m) \end{aligned} \quad (4.14)$$

An (4.14) kann die Zunahme der Laufzeit in Abhängigkeit von den Eigenschaften n und m der Trainingsmenge und von allen Parametern abgelesen werden. Wenn alle von n und m unabhängigen Parameter als Konstanten im Sinne der O-Notation aufgefaßt und die beiden von n abhängigen Parameter *N_CONDS* und *ATTS_MAX* (vgl. 4.2.7) durch n ersetzt werden, ergibt sich folgendes Ergebnis in n und m:

$$C_{LARGE} = O(n^4 + n^2 m \log m) \quad (4.15)$$

Es ist festzustellen, daß der Aufwand polynomiell ist, die Anzahl der Eingabeattribute mit höchstens n^4 und die Größe der Trainingsmenge insbesondere mit höchstens $m \log m$ in den Aufwand eingeht. Der eine der beiden n^4-Terme aus (4.14) gilt, wie erörtert, auch nur für den „schlechten" Fall, daß viele Argumentkombinationen für neue Attribute durch die Bildungsregeln abgelehnt werden – im besten Fall eines Erfolgs der höchstkorrelierten Attribute reduziert sich dieser Summand auf n^2.

4.2.9 Anwendungsbeispiele

In diesem Abschnitt werden einige Beispiele für die Anwendung von *LARGE* als isoliertem Lernverfahren vorgestellt – unabhängig vom Problem des Protokollernens.

Dazu wurden Testdaten nach vorher festgelegten Regeln generiert und als Eingabe für *LARGE* verwendet, wobei dem Algorithmus natürlich keinerlei Informationen über die Erzeugungsregeln zur Verfügung standen. Insbesondere ist ihm die Zahl der Klassen unter den Eingabedaten, also die Anzahl der im Idealfall in der Hypothese erwarteten Klauseln, nicht vorgegeben. Die Lehrbeispiele verschiedener Klassen sind in der Trainingsmenge zufällig miteinander vermischt.

Datensatz 1

Figur A.42 zeigt im linken Bildteil die Regeln zur Datenerzeugung für den ersten Beispieldatensatz. Die Regeln bestehen für jede der zwei Klassen aus je vier Gleichungen, von denen jede ein Eingabeattribut $v_1, \ldots, v_4$ festlegt. Die Intervallnotation $a..b$ steht für eine gleichverteilte zufällige Ziehung einer Zahl aus dem angegebenen Intervall, die restlichen Komponenten umfassen die Verwendung anderer, schon bestimmter Eingabeattribute und arithmetische Operationen. Wenn statt einer Bestimmungsgleichung eine Gleichungskette angegeben ist, dient alles ab dem zweiten Gleichheitszeichen nur noch der kommentierenden Beschreibung zur besseren Übersicht – in diesem Fall ist die Gleichverteilung bei Intervallangaben nicht mehr unbedingt gegeben, weil mehrere zufällige Ziehungen überlagert sein können.

Im rechten Teil der Figur ist das Lernergebnis zu sehen. Die abgeleiteten Attribute sind hier in Termnotation angegeben, mehrere Bedingungen einer Klausel trennt ein *AND*. In eckigen Klammern erscheint unter jeder Klausel zunächst die Zahl der Klauseltreffer, also die Mächtigkeit der von dieser Klausel beschriebenen Klasse innerhalb der Trainingsmenge. Hinter dem ersten Schrägstrich steht die Anzahl der Diskriminationen, also der „Verkleinerungen" der Treffermenge (vgl. 4.2.7). Der dritte Zahlenwert ist die Klauselqualität gemäß Q. Man kann die Bewertung der Termkomplexität in den Einer- und Zehnerstellen der Klauselqualität nachvollziehen, die Komponente *compl* aus (4.6): Bei beiden Klauseln beträgt die mittlere Anzahl Operatoren pro Bedingungsterm $5/5 = 1,0$, so daß $compl = 99 - 10 \cdot 1,0 = 89$.

Die Lernaufgabe zeigt als besondere Schwierigkeit, daß die beiden Klassen im Attribut v_1 überlappen und im Attribut v_2 mit etwa gleicher Beispieldichte direkt nebeneinander liegen. Trotzdem werden beide Klassen exakt diskriminiert. Die höhere Klauselqualität bei der ersten Klasse entsteht durch die bessere Auffälligkeitsbewertung der beiden ersten Einzelwertbedingungen und die Ablehnung *aller* Beispiele der anderen Klasse durch diese beiden Bedingungen. Man erkennt, daß die Klauseln sowohl *diskriminierende* als auch *deskriptive* Bedingungen enthalten: In beiden Klauseln genügt die jeweils erste Bedingung zur Klassendiskrimination. Die weiteren Bedingungen hat *LARGE* zur weitergehenden Charakterisierung der jeweiligen Klasse erzeugt. Die Bedingungen sind nicht völlig frei von Redundanz, beispielsweise folgt die dritte Bedingung $-4 \leq v_3 - v_2 \leq 0$ in der ersten Klausel aus den Bedingungen zwei, $v_4 - v_2 + v_3 = 0$, und fünf, $0 \leq v_4 \leq 4$. Dies ist möglich, weil nur die Regeln nach Abschnitt 4.2.6 und kein vollständiger Ungleichungsumformer zur Vermeidung redundanter Regeln eingesetzt werden. Immerhin verhindern die Bildungsregeln die Ableitung paarweise äquivalenter Bedingungen.

Ein Versuch mit denselben Klassen, aber 300 Beispielen im gleichen Verhältnis liefert im wesentlichen das gleiche Ergebnis, dargestellt in Figur A.43.

Datensatz 2

Figur A.44 zeigt die zweite Lernaufgabe, diesmal sogar mit vier Klassen. Um die „Sicht" des Algorithmus auf die Lernaufgabe noch besser zu illustrieren, befindet sich in der Figur auch eine Tabelle mit einem Auszug aus der Trainingsmenge, wie *LARGE* sie als Eingabe erhält.

Auch hier werden alle charakteristischen Gleichungen gefunden. Gut kann man die tatsächlichen Extremwerte der zufällig gezogenen Attributwerte innerhalb der Trainingsmenge ablesen. Bei den beschreibenden Attributen ergeben sich zahlreiche Aussagen über die Summen der Zufallswerte, die zwar keine charakteristischen Eigenschaften der vorgegebenen Klassen sind, aber natürlich zutreffende Aussagen über die Lehrbeispiele. Eine klarere Identifikation der *Idee* hinter der Klassenbeschreibung kann von einem Suchalgorithmus auch kaum erwartet werden.

Tabelle 4.3 zeigt ein Protokoll der Lernschritte bis zur Ausgabe der Hypothese für diese Lernaufgabe durch *LARGE*. Die Spalten der Tabelle bezeichnen die Zahl der Klauseln in C, die Zahl der Bedingungen in allen Klauseln, die Zahl der abgeleiteten Attribute (Terme), die Zahl der bereits akzeptierten Beispiele, die Gesamtzahl der Klauseltreffer einschließlich Überschneidungen, die aktuelle Länge der Teilmengen-Tabuliste sowie die Qualität Q^* der letzten Auswahl. *Auswahl* in der ersten Spalte kennzeichnet eine neue beste Auswahl, also eine neue Hypothese.

Datensatz 3

Figur A.45 stellt eine Lernaufgabe mit nur einer Klasse in der Trainingsmenge dar. *LARGE* erzeugt tatsächlich eine Hypothese mit nur einer Klausel, weil alle weiteren erzeugten Klauseln von der Klauselqualität schlechter bewertet sind und auch keine überlappungsfreie Klassifikation ergeben.

Datensatz 4

Das vierte Beispiel, zu sehen in Figur A.46, ist dahingehend besonders, daß zur Diskrimination der vier Klassen jeweils *zwei* Bedingungen erforderlich sind. Im Diagramm links unter der Datenbeschreibung sieht man, warum das so ist. Auch diese Diskriminationsregeln erkennt *LARGE* durch iterative Isolation von Häufungen der Attributwerte.

Laufzeit

Nach den inhaltlichen Ergebnissen der vorangegangenen Unterabschnitte erfolgt jetzt noch eine Zusammenstellung der zugehörigen Laufzeiten. Die Zeiten wurden mit der *LARGE*-Testimplementierung unter Linux 2.0 auf einem PentiumPro-basierten Rechner mit 200 MHz Takt gemessen. Tabelle 4.4 stellt die Ergebnisse dar. Man erkennt, daß die mit $O(m \log m)$ abgeschätzte Abhängigkeit von der Größe der Trainingsmenge bei diesen Beispielen noch hinter einer linearen Zunahme zurückbleibt.

Symbol	Bedeutung
n	Anzahl der Attribute $v_1, \ldots, v_n$ pro Lehrbeispiel (Eingabeattribute).
m	Anzahl der Lehrbeispiele, $m = \|E\|$.
c	Anzahl der schon vorhandenen Klauseln in C.

Tabelle 4.2: Freie Variablen in den Termen für den Laufzeitaufwand von *LARGE*.

Schritt	Klau-seln	Bedin-gungen	Terme	Abdek-kung	Treffer	Tabus	Q^*
1	1	4	4	25	25	1	—
2	2	8	7	30	30	2	—
3	3	12	8	50	55	3	—
4	4	16	11	74	79	4	—
5	5	20	14	99	178	4	—
6	6	24	14	99	203	4	—
7	6	24	14	99	203	5	—
8	7	28	17	99	227	4	—
9	8	32	17	100	252	4	664
Auswahl	4	16	17	100	100	4	664
10	9	34	17	100	278	4	652
11	10	38	17	100	293	4	646
12	10	38	17	100	293	5	646
13	10	38	18	100	199	4	643
14	10	38	15	100	199	5	643
15	10	38	16	100	199	6	643

Tabelle 4.3: Lernprotokoll für Datensatz 2.

Datensatz	n	m	Schritte	Zeit [s]	Zeit/Schritt [s]
1	4	30	9	1,03	0,114
1	4	300	11	6,95	0,632
1	4	900	8	12,91	1,614
2	3	100	15	1,28	0,085
2	3	1000	13	7,43	0,572
3	4	100	10	2,41	0,241
4	2	400	10	0,22	0,022

Tabelle 4.4: Laufzeiten von *LARGE* bei den Lernbeispielen.

4.3 Lernen des PDU-Typen-Automaten

Dieser Abschnitt behandelt den anderen Teil der Protokoll-Lernaufgabe, den endlichen Automaten der PDU-Typen. Die folgenden Unterabschnitte geben noch einmal kurz die Spezifikation dieser Teilaufgabe an und beschreiben dann erst den Lösungsansatz, die aufwandsoptimierte Form des Algorithmus und schließlich die Art der erzeugten Automaten. Dieser Algorithmus heißt *InfSM*-Algorithmus (für ***infer a [finite] state machine***, Ableitung eines endlichen Automaten).

4.3.1 Spezifikation

Die Ein-Ausgabe-Relation von *InfSM* läßt sich unmittelbar aus der Spezifikation der vollständigen Protokoll-Lernaufgabe in Abschnitt 4.1.1 ablesen.

Eingabe: Eine Wort $\alpha = a_1 \dots a_n \in \Sigma^*$ aus n PDU-Typen, die beobachtete PDU-Sequenz ohne den Datenanteil.

Ausgabe: Die Zustandsmenge S und die Transitionsrelation $\delta \subseteq S \times \Sigma \times S$ des zu lernenden Protokollautomaten A (vgl. Abschnitt 4.1.1). S und δ müssen α akzeptieren, also:

$$\exists r_0, \dots, r_n \in S \quad (r_0, a_1, r_1), (r_1, a_2, r_2), \dots, (r_{n-1}, a_n, r_n) \in \delta \tag{4.16}$$

4.3.2 Ansatz

Das Kernproblem bei der Konstruktion eines Automaten, der α akzeptiert, besteht darin, die Positionen i zwischen zwei aufeinanderfolgenden Symbolen $a_i a_{i+1}$ in der Folge α mit Zuständen $r_i \in S$ zu identifizieren, wobei S selbst ja noch zu bilden ist. Die PDU-Typen-Folge gibt keine direkte Auskunft über Kommunikationszustände, denn diese Zustände sind ein Konzept des Protokollentwurfs, ergeben sich also aus der Intention des Protokollentwicklers im Hinblick auf die Protokollsemantik. Dem *InfSM*-Algorithmus bleibt als einziger Ansatzpunkt, Teilsequenzen von α miteinander zu vergleichen oder sonst irgendwie auf ihnen zu operieren.

Eine sehr vorteilhafte Sicht für die Automatenkonstruktion ist die, daß jeder Zustand $r \in S$ gerade eine Äquivalenzklasse von Positionen in der PDU-Sequenz α darstellt. Für die gesamte Zustandsmenge S bedeutet dies das folgende: Es wird eine Äquivalenzrelation

$$\sim \subseteq \{0; \ldots; n\}^2$$

eingeführt, so daß S als der Faktorraum von $\sim$ identifiziert werden kann und $\sim$ gerade die r_i aus (4.16) definiert:

$$\begin{aligned} S \stackrel{def}{=} \{0; \ldots; n\}/\sim \quad &= \quad \{[i] \mid i \in \{0; \ldots; n\}\} \\ \forall i, j \in \{0; \ldots; n\} \quad & r_i = r_j \Leftrightarrow i \sim j \end{aligned} \tag{4.17}$$

Wenn alle Zustände r_i in dieser Weise identifiziert sind, kann die Transitionsrelation δ trivial aus $\sim$ und α abgeleitet werden, so daß (4.16) automatisch erfüllt ist:

$$\delta \stackrel{def}{=} \{([i-1], a_i, [i]) \mid i \in \{1; \ldots; n\}\} \tag{4.18}$$

δ enthält also alle Zustandsübergänge zwischen Zuständen, die gemäß $\sim$ in α aufeinander folgen, mit dem zwischen diesen beiden Positionen in α stehenden Symbol als Kommunikationsprimitive. Die denkbaren Extremfälle sind:

1. Der universelle Akzeptor mit einem einzigen Zustand (vgl. Figur A.38):

$$\begin{aligned} \sim &= \{0; \ldots; n\}^2 \\ S &= \{[0]\} \\ \delta &= S \times \Sigma \times S \end{aligned}$$

2. Der auf α beschränkte Akzeptor mit $n+1$ Zuständen (vgl. Figur A.37):

$$\begin{aligned} \sim &= \{(0,0); (1,1); \ldots; (n,n)\} \\ S &= \{\{0\}; \{1\}; \ldots; \{n\}\} \\ \delta &= \{(\{0\}, a_0; \{1\}); \ldots; (\{n-1\}, a_n, \{n\})\} \end{aligned}$$

Bleibt zu klären, wie sich eine Äquivalenzrelation $\sim$ sinnvoll und effizient aus α ableiten läßt.

Da $\sim$ *ähnliche* Positionen in der PDU-Folge zusammenfaßt, ist sie auf der Grundlage eines Ähnlichkeitsmaßes zu definieren. Für die Ähnlichkeit zweier α-Positionen, d. h. für ihre Zugehörigkeit zum selben Automatenzustand, spricht, wenn sie Gemeinsamkeiten in der unmittelbar vorausgegangenen oder unmittelbar nachfolgenden Teilsequenz von Symbolen aufweisen. Dies wird im folgenden als gemeinsames *Präfix* bzw. gemeinsames *Postfix* bezeichnet. Wenn zwei α-Positionen i und j ein besonders langes gemeinsames Präfix haben, so ist es wahrscheinlich, daß an diesen beiden Stellen derselbe Kommunikationszustand vorlag. i und j können allerdings völlig verschiedene Postfixe zeigen, wenn der Protokollautomat unterschiedliche Folgezustände angenommen hat. i könnte aber z. B. im Postfix eine hohe Übereinstimmung mit einer weiteren Position k zeigen und j mit einer Position l. Wenn jetzt

noch k und l ein langes gemeinsames Präfix tragen, ist es wahrscheinlich, daß alle vier Positionen zur selben Äquivalenzklasse, also zum selben Kommunikationszustand, gehören sollten. Figur A.47 zeigt ein konkreteres Beispiel für eine derartige Situation.

Diese Überlegungen deuten an, daß die zustandsbildenden Äquivalenzklassen durch den transitiven Abschluß einer Ähnlichkeitsrelation entstehen sollten. Die Ähnlichkeitsrelation ist zu definieren über ein Ähnlichkeitsmaß auf den Sequenzpositionen, das sich aus der Länge gemeinsamer Prä- und Postfixe berechnet.

Dieses Ähnlichkeitsmaß heiße pp_α (für Präfix und Postfix) und wird folgendermaßen definiert:

$$\begin{aligned} pp_\alpha &: \{0;\ldots;|\alpha|\}^2 \to \mathbb{N} \\ pp_\alpha(i,j) &\stackrel{def}{=} pp^-_\alpha(i,j) + pp^+_\alpha(i,j) \\ &\text{mit} \\ pp^-_\alpha(i,j) &\stackrel{def}{=} \max\left\{z \mid a_{i-z+1}\ldots a_i = a_{j-z+1}\ldots a_j\right\} \\ pp^+_\alpha(i,j) &\stackrel{def}{=} \max\left\{z \mid a_{i+1}\ldots a_{i+z} = a_{j+1}\ldots a_{j+z}\right\} \end{aligned} \tag{4.19}$$

Die Summe aus der Länge gemeinsamer Prä- und Postfixe entspricht natürlich der größten Länge einer Teilsequenz, innerhalb der oder an deren Ende die verglichenen Positionen liegen.

Aus dem Ähnlichkeitsmaß pp wird die Ähnlichkeitsrelation $\dot{\sim}_N$, indem man eine Schwelle $N \in \{0;\ldots;n\}$ für die geforderte Ähnlichkeitsbewertung festlegt:

$$\begin{aligned} \dot{\sim}_N &\subseteq \{0;\ldots;n\}^2 \\ i \dot{\sim}_N j &\Leftrightarrow pp_\alpha(i,j) \geq N \end{aligned} \tag{4.20}$$

Diese Ähnlichkeitsrelation ist aufgrund (4.19) zwar symmetrisch und reflexiv, aber nicht unbedingt transitiv und somit noch keine Äquivalenzrelation. Um eine schwellenabhängige Zustandsrelation $\sim_N$ zu erhalten, müßte die transitive Hülle von $\dot{\sim}_N$ gebildet werden:

$$\sim_N \stackrel{def}{=} \mathrm{t}(\dot{\sim}_N)$$

InfSM macht diesen Schritt jedoch nicht direkt. Die Anzahl der gebildeten Äquivalenzklassen, also Zustände, wächst zwar monoton mit der Schwelle N, läßt sich aber ohne die Berechnung der transitiven Hülle kaum anhand der Schwelle abschätzen. Daher soll *InfSM* gleich eine Ähnlichkeitsmatrix *sim* berechnen, aus der *alle* $\sim_N$ für sämtliche Schwellen unmittelbar ablesbar sind:

$$\begin{aligned} sim &: \{0;\ldots;n\}^2 \to \mathbb{N} \\ sim(i,j) \geq N &\Leftrightarrow i \sim_N j \end{aligned} \tag{4.21}$$

Im Moment ist noch nicht völlig klar, ob (4.21) eine widerspruchsfreie und eindeutige Festlegung von *sim* darstellt. Die Kodierung aller Äquivalenzrelationen in *sim* ist zunächst nur eine Absicht.

sim soll wie folgt berechnet werden:

1. Setze $sim_0 = pp_\alpha$.
2. Bilde sim_{z+1} durch

$$sim_{z+1}(i,j) = \max_{0 \le k \le n} \min\{sim_z(i,k); sim_z(k,j)\} \tag{4.22}$$

   solange, bis $sim_{z+1} = sim_z$.
3. sim ist gleich sim_{z+1}.

Die Operation in Schritt 2 entspricht übrigens genau dem Quadrat von sim_z als Matrix, wenn man die Multiplikation durch min und die Addition durch max ersetzt. Dies erlaubt eine verkürzte Notation:

$$sim = \lim_{z \to \infty} (pp_\alpha)^z$$

Theorem 7 *Der Bildungalgorithmus für sim nach (4.22) terminiert. Die Interpretation von sim gemäß (4.21) liefert für jede Schwelle N eine Äquivalenzrelation.*

Beweis: Da die Hauptdiagonaleneinträge nach der Definition von pp (4.19) maximal in ihrer Zeile und Spalte sind, wird die rechte Seite von (4.22) durch den Fall $k = i$ nie kleiner als

$$\min\{sim_z(i,i); sim_z(i,j)\} = sim_z(i,j),$$

die stets ganzzahligen Matrixelemente können also nur zunehmen, sind durch das globale Maximum in pp_α aber nach oben beschränkt. Daraus folgt die Termination nach endlich vielen Iterationsschritten.

Die in sim kodierten Relationen sind symmetrisch wegen der Symmetrie von pp und der Vorschrift (4.22). Sie sind reflexiv, da die Elemente der Hauptdiagonale größer oder gleich der größten Schwelle $N = n$ sind. Die Transitivität wird durch Widerspruch gezeigt: Angenommen, es existieren i, j, k, N, so daß

$$sim(i,k) \ge N \quad \wedge \quad sim(k,j) \ge N \quad \wedge \quad sim(i,j) < N.$$

Dann wäre im letzten Schritt nach (4.22)

$$sim_{z+1}(i,j) \ \ge \ \min\{sim_z(i,k); sim_z(k,j)\} \ \ge \ N \ > \ sim_z(i,j),$$

was einen Widerspruch zur Abbruchbedingung darstellt. □

Daß sim tatsächlich die transitiven Hüllen der $\dot{\sim}_N$ kodiert, ergibt ein direkter Vergleich des Algorithmus (4.22) mit dem trivialen Verfahren zur Berechnung der transitiven Hülle für ein festes N: Genau dann, wenn ein Element k in Relation zu i und j steht, wird auch die Äquivalenz zwischen i und j eingetragen.

Nach der Berechnung von sim können die resultierenden S und δ für jede Schwelle N abgelesen werden. Dies erleichtert die Wahl von N anhand einer Abschätzung der Zustandszahl, die eher a priori möglich ist als eine Vorgabe der Ähnlichkeitsschwelle. Figur A.48 zeigt ein Beispiel für die Veränderung eines abgeleiteten Automaten durch Einführung einer zusätzlichen Äquivalenz zwischen zwei Zuständen des ersteren.

4.3.3 Optimierter Algorithmus

Nach den bisherigen Angaben könnte *InfSM* schon direkt implementiert werden. Wegen der modifizierten Matrixmultiplikation, die iterativ durchgeführt wird, ergäbe dies jedoch einen Laufzeitaufwand im schlimmsten Fall in der Größenordnung $O(n^4)$, was nicht akzeptabel ist. Im folgenden wird der *InfSM*-Algorithmus mit dem oben spezifizierten Ergebnis, aber einem Laufzeitaufwand von höchstens $O(n^2)$ formuliert:

Prozedur *InfSM*(α)
$\alpha = a_1 \dots a_n \ : \Sigma^*$

-- Berechnung des Ähnlichkeitsmaßes pp
Für $d \in 0 \dots n$
$pp(0,d), pp(d,0) \leftarrow 0$
$c \leftarrow 0$
Für $i \in 1 \dots n-d$ -- Präfixe
Wenn $a_i = a_{i+d}$ **Dann** $c \leftarrow c+1$ **Sonst** $c \leftarrow 0$
$pp(i, i+d), pp(i+d, i) \leftarrow c$
$c \leftarrow 0$
Für $i \in d+1 \dots n$ Absteigend -- Postfixe
Wenn $a_{i-d} = a_i$ **Dann** $c \leftarrow c+1$ **Sonst** $c \leftarrow 0$
$c' \leftarrow pp(i-d-1, i-1)$
$pp(i-d-1, i-1), pp(i-1, i-d-1) \leftarrow c + c'$

-- Erzeugung der Klassenzuordnung M
$L \leftarrow \{pp(i,j) | i,j \in 0 \dots n\}$ -- alle Schwellen
$\langle \lambda(N,i) \rangle_{N \in L, i \in 0 \dots n} \leftarrow \langle \text{list}\{j \mid pp(i,j) = N\} \rangle$ -- Matrix von Kantenlisten
$\langle \lambda'(N,i) \rangle_{N \in L, i \in 0 \dots n} \leftarrow \langle \emptyset \rangle$ -- Hilfsmatrix

$M \leftarrow \text{id}_{0 \dots n}$
Für $N \in L$ Absteigend
$X \leftarrow \text{range}(M)$ -- bisherige Repräsentanten
$M_N \leftarrow \text{id}_X$
Für $i \in 0 \dots n$ -- Kanten auf Repräsentanten umsetzen
Für $j \in \lambda(N,i)$
$\lambda'(N, M(i)).\text{append}(M(j))$
Für $i \in 0 \dots n$ **Wenn** $i \in X$ -- Repräsentant von $[i]$
$Y \leftarrow \{i\}$
Solange $Y \neq \emptyset$ **Wiederhole** -- $[i]$ traversieren
Wähle $j \in Y$ -- j wird expandiert
$Y \leftarrow Y \setminus \{j\}$
$M_N(j) \leftarrow i$ -- Zuordnung $j \in [i]$ eintragen
Für $k \in \lambda'(N,j)$ -- j durch unbesuchte Nachbarn ersetzen
Wenn $k \in X$
$Y \leftarrow Y \cup \{k\}$

$X \leftarrow X \setminus \{k\}$
$M_N, M \leftarrow M_N \circ M$ -- mit bisheriger Zuordnung verketten

-- Bildung des Automaten
Wähle N in Abhängigkeit von $|\text{range}(M_N)|$
$S \leftarrow \text{range}(M_N)$
$\delta \leftarrow \{(M_N(i-1), a_i, M_N(i)) \mid i \in 1 \ldots n\}$
Ausgabe S, δ

Ende von *InfSM*

Statt der Ähnlichkeitsmatrix *sim* berechnet der dargestellte Algorithmus die Äquivalenzklassen-Zuordnungsfunktionen M_N, die zu jeder Sequenzposition bereits ein Kennzeichen der Äquivalenzklasse, also des Zustands, liefern:

$$M_N : \{0; \ldots; n\} \rightarrow \{0; \ldots; n\} \qquad (0 \leq N \leq n)$$
$$M_N(i) = M_N(j) \Leftrightarrow sim(i,j) \geq N$$

Kern des Algorithmus ist eine Schleife über die sinnvollen Schwellenwerte, in der die Zustandsklassen *iterativ* konstruiert werden, d. h. aus der durch M_N beschriebenen Zustandsmenge der Relation $\sim_N$, in die alle Ähnlichkeiten von N und darüber eingegangen sind, wird im nächsten Schritt die Zustandsmenge der Relation $\sim_{N-1}$ berechnet und in M_{N-1} kodiert. Durch die Größer-oder-gleich-Bedingung in (4.21) kann es dabei keine Änderungen außer der Zusammenfassung von Klassen geben.

Die Wahl einer angemessenen Ähnlichkeitsschwelle N wird weiter unter im Rahmen des Erlernens realer Protokollregeln genauer behandelt.

4.3.4 Aufwand

Dieser Abschnitt erläutert, warum *InfSM* in der dargestellten Form eine Laufzeit von nicht mehr als $O(n^2)$ in der Länge der Eingabesequenz benötigt:

- Bei der Berechnung des Ähnlichkeitsmaßes *pp* wird ausgenutzt, daß eine mehrfach auftretende Teilsequenz die Prä- und Postfixlängen für mehrere Matrixeinträge in *pp* beeinflußt. Die Symbolfolge α wird für jeden Vergleichsabstand d je einmal vorwärts und rückwärts durchlaufen, wobei die Matrix *pp* in diagonalen Streifen ausgefüllt wird. Figur A.49 veranschaulicht diese Vorgehensweise. Weil die Schachtelungstiefe der Schleifen bei zwei liegt, ergibt sich ein Aufwand von höchstens $O(n^2)$.

- Ein wichtiger Schritt ist die anschließende Umordnung der vorkommenden Ähnlichkeitswerte in die Matrix λ. Anschließend ist $\lambda(N,i)$ die *Liste* aller Knotennummern, die mit Knoten i die Ähnlichkeit N haben ($pp(i,j) = N$). Da nur $(n+1)^2$ Indizes einzutragen sind, die höchste Ähnlichkeit n beträgt und das Anfügen an eine Liste ohne weiteres mit konstantem Aufwand möglich ist, ergibt sich wiederum $O(n^2)$ Zeitbedarf.

- Der erste Teilschritt der Klassenbildung ist die Berechnung der Kantenlistenmatrix λ', die dieselbe Struktur wie λ aufweist, in der aber alle Kanten mit einer Ähnlichkeit von genau N auf die Klassenrepräsentanten der letzten Zerlegung M „umgehängt" werden: Alle Kanten von und zu einer M-Klasse werden dem Repräsentantenknoten zugeordnet. Für Repräsentantenknoten i gilt $M(i) = i$. Die *append*-Operation muß für den gesamten Algorithmus höchstens $(n+1)^2$-mal ausgeführt werden, weil nur so viele Kanten in λ vorhanden sind, Aufwand also $O(n^2)$.

  Figur A.50 stellt die Folge der Teilschritte für den Übergang von $N = 3$ zu $N = 2$ beispielhaft dar. In dieser Figur sind Repräsentantenknoten ausgefüllt gezeichnet, während die restlichen Knoten j einer Klasse mit $M(j)$, dem Index des Repräsentantenknotens als Kennzeichen ihrer Klasse, markiert sind.

- Nun wird nacheinander von jedem in X vermerkten Repräsentantenknoten der Zerlegung M jeweils die zugehörige Äquivalenzklasse entlang der Kanten mit $pp(i,j) = N$ traversiert. Knotenindex i wird zum Kennzeichen dieser Klasse. Y dient als Knotenspeicher für die verwendete Breitensuche. Entscheidend für den Aufwand dieses Teils ist der Rumpf der innersten Schleife (Für $k \ldots$): Durch die Löschung der schon besuchten Knoten aus X wird pro Ähnlichkeitsschwelle N kein Knoten j doppelt expandiert. Insgesamt gibt es in λ' aber nur $(n+1)^2$ Kanten, so daß der innerste Schleifenrumpf auch nur so oft durchlaufen werden kann und der Aufwand durch $O(n^2)$ beschränkt bleibt.

- M_N enthält zunächst nur die neuen Zusammenfassungen von M-Repräsentanten gemäß N-Kanten. Die resultierende Zerlegung mit korrekten Klassenkennzeichen für *alle* Knoten erhält man erst durch die Komposition $M_N \circ M$ (siehe auch Figur A.50). Die $\circ$-Operation benötigt $n+1$ Elementaroperationen, weil M auf so vielen Stellen definiert ist, und wird in der äußersten Schleife höchstens n-mal ausgeführt. Daraus folgt ein Zeitaufwand von $O(n^2)$.

- Der kritische Aspekt sind die Operationen auf den Knotenmengen X bzw. Y: Die Löschoperationen aus X müssen offenbar mit konstantem Aufwand erfolgen, damit die angegebene Aufwandsabschätzung gilt. Dies ist leicht möglich, wenn X z. B mit Hilfe eines Bitvektors repräsentiert wird. Aus diesem Grund wurde eine Schleife der Form „Solange $X \neq \emptyset$" vermieden, denn dieser Test ist bei einem Bitvektor *nicht* mit konstantem Aufwand durchführbar. Statt dessen wurde die Schleife über die Mitglieder von X als Für-Schleife mit explizitem Mitgliedstest notiert – was natürlich dazu führt, daß bereits deren Rumpf $O(n^2)$-mal ausgeführt wird. Die innere Schleife über die Mitglieder von Y darf pro Element nun nur noch konstanten Aufwand haben, was man mit einer Repräsentation von Y z. B. als verzeigerter Keller erreichen kann. Auf Y finden nur der Test auf die leere Menge, das Einfügen eines bisher nicht enthaltenen Elements sowie die kombinierte Auswahl und Löschung eines beliebigen Elements statt. Alle diese Operationen verursachen in einer üblichen Kellerimplementierung nur konstanten Aufwand.

Da alle beschriebenen Teilaufwände additiv zum Gesamtaufwand von *InfSM* zusammengefaßt werden, ist auch dieser schlimmstenfalls quadratisch:

$$C_{InfSM} = O(n^2) \tag{4.23}$$

4.3.5 Eigenschaften des gelernten Automaten

Die zustandsbildende Äquivalenzrelation zwischen den Sequenzpositionen als Länge gemeinsamer umgebender Teilsequenzen verkörpert eine recht intuitive Definition der Automatenzustände. Nun stellt sich die Frage nach Eigenschaften der auf diese Weise automatisch erzeugten endlichen Automaten.

Aus der Automatentheorie ist bekannt, daß deterministische endliche Automaten (DEA) als Spezialfälle nichtdeterministischer endlicher Automaten (NEA) die gleiche Mächtigkeit hinsichtlich der modellierbaren Sprachen aufweisen, daß also jeder endliche Automat in einen äquivalenten DEA überführt werden kann. Der wesentliche Vorteil eines DEA ist, daß von einem bekannten Zwischenzustand aus der jeweilige Folgezustand stets eindeutig durch das verarbeitete Symbol bestimmt wird. Formal sind DEAs gegenüber äquivalenten NEAs dadurch ausgezeichnet, daß der konstruktive Algorithmus zur Umformung eines NEA in einen äquivalenten DEA aus k Zuständen des NEA bis zu 2^k Zustände des DEA macht, nämlich gerade die Potenzmenge des ursprünglichen Zustandsraums.

Die von *InfSM* erzeugten endlichen Automaten sind im allgemeinen NEAs. Die folgende Betrachtung des Algorithmus soll jedoch zeigen, daß *InfSM* „häufig" deterministische Transitionen konstruiert. Ein gutes Maß dafür, „wie" nichtdeterministisch ein gegebener NEA ist, bildet die Zahl der Zustands-Symbol-Paare (r, a), für die mehr als ein Tripel $(r, a, r') \in \delta$ existiert. Es soll daher eine *notwendige Bedingung* dafür ermittelt werden, daß (r, a) ein nichtdeterministisches Zustands-Symbol-Paar in der Transitionsrelation des mit *InfSM* erzeugten NEA A bildet.

Angenommen, (r, a) ist ein solches nichtdeterministisches Zustands-Symbol-Paar. Dann existieren in der erzeugten Relation δ zwei Tripel

$$(r, a, r_1), (r, a, r_2) \in \delta \qquad (r, r_1, r_2 \in S,\ r_1 \neq r_2, a \in \Sigma)$$

oder, bezogen auf die Eingabesequenz α und die Positionszerlegung M mit $\text{range}(M) = S$,

$$(M(i-1), a_i, M(i)) \in \delta,\ (M(i-1), a_i, M(j)) = (M(j-1), a_j, M(j)) \in \delta,\ M(i) \neq M(j).$$

Wegen $M(i-1) = M(j-1)$ und weil M die Sequenzpositionen in Äquivalenzklassen zerlegt, existiert eine Folge $\langle \mu_k \rangle_{k \in 1 \ldots z}$ von Sequenzpositionen, so daß

$$pp(i-1, \mu_1) \geq N \wedge pp(\mu_1, \mu_2) \geq N \wedge \ldots \wedge pp(\mu_{z-1}, \mu_z) \geq N \wedge pp(\mu_z, j-1) \geq N,$$

wobei N die gewählte Ähnlichkeitsschwelle ist. Ferner muß $sim(i, j) < N$ gelten, sonst wäre $i \sim_N j$, und die Folgezustände r_1 und r_2 wären von *InfSM* zum selben Zustand zusammengefaßt worden.

Wegen $a_i = a_j$ gilt aber:

$$
\begin{aligned}
pp^-(i, \mu_1 + 1) &= pp^-(i-1, \mu_1) + 1 \quad \text{(Präfixlänge)} \\
pp^+(i, \mu_1 + 1) &= pp^+(i-1, \mu_1) - 1 \quad \text{(Postfixlänge)} \\
pp^-(j, \mu_z + 1) &= pp^-(j-1, \mu_z) + 1 \quad \text{(Präfixlänge)} \\
pp^+(j, \mu_z + 1) &= pp^+(j-1, \mu_z) - 1 \quad \text{(Postfixlänge)} \\
\Rightarrow & \\
pp(i, \mu_1 + 1) &= pp(i-1, \mu_1) \geq N \\
pp(j, \mu_z + 1) &= pp(j-1, \mu_z) \geq N
\end{aligned}
$$

Wenn nun aber $z = 1$, so folgt daraus $sim(i,j) \geq N$, also doch $sim(i,j) \geq N$ und mithin $r_1 = M(i) = M(j) = r_2$, Widerspruch! Es läßt sich also formulieren:

> Wenn $(r, a, r_1), (r, a, r_2)$ zwei nichtdeterministische Übergänge zum Zustands-Symbol-Paar (r, a) sind, dann kann es zwischen den zugehörigen Sequenzpositionen i, j mit $r = M(i) = M(j)$ nur Pfade mit *mindestens zwei Zwischenpositionen* geben, die i und j über die Ähnlichkeitsrelation $\sim_N$ – vor Bildung der transitiven Hülle – verbinden.

Praktisch bedeutet dies aber, daß bereits eine hinreichend vollständige Eingabesequenz α einen DEA als Lernergebnis gewährleistet: Wenn α von einem DEA A_0 erzeugt wurde und für jeden Zustand q von A_0 sämtliche Kombinationen der möglichen N-stelligen Präfixe und Postfixe zu q mindestens einmal in α vorkommen, so erfolgt die Bildung der transitiven Hülle von $\sim_N$ grundsätzlich über höchstens *eine* Zwischenposition – der gelernte Automat A ist dann deterministisch. Die praktischen Ergebnisse am Ende dieses Kapitels werden zeigen, wie gut dieser Idealfall in der Realität approximiert wird.

4.3.6 Bewertung

Hier sollen die theoretischen Ergebnisse zu *InfSM* in einen Zusammenhang gestellt werden.

Die Ähnlichkeitsrelation liefert eine pro Schwellenwert *eindeutige* Zerlegung in Äquivalenzklassen, mithin einen eindeutig definierten endlichen Automaten.

Den Zustand eines endlichen Automaten als die Äquivalenzklasse der ähnlichsten Vor- und Folgeverhalten zu interpretieren, erscheint als eine natürliche und globale, wohldefinierte und leicht nachvollziehbare Sicht auf das Konzept des Zustands. Es spricht einiges dafür, die nach diesem Kriterium erhaltenen Automaten hinsichtlich der Aussagekraft ihrer Zustände als *optimal* einzustufen.

Modulo dieses Optimalitätsbegriffs läßt sich daher zusammenfassen, daß *InfSM* – mit einem Aufwand von $O(n^2)$ in der Länge der Eingabesequenz – *alle optimalen* endlichen Automaten findet, die die Eingabesequenz akzeptieren. Unabhängig vom *InfSM*-Algorithmus gilt also ganz allgemein, daß jedes Wort eine Familie optimaler NEAs *definiert*, die sich paarweise in der Zahl ihrer Zustände unterscheiden. Bei einer günstiger Eingabesequenz werden sogar deterministische Automaten konstruiert.

4.4 Protokollregelerwerb

Dieser Abschnitt beschreibt, wie die oben entwickelten Algorithmen *LARGE* und *InfSM* gemeinsam zur Lösung der Protokoll-Lernaufgabe engesetzt werden können.

Zunächst wird angegeben, wie man *LARGE* und *InfSM* in einen konkreten Lernalgorithmus integriert. Weitere Unterabschnitte thematisieren die Behandlung sehr großer Beispielverhalten und die Kontrolle der Zustandszahl, die *InfSM* in Abhängigkeit von der Ähnlichkeitsschwelle liefert.

4.4.1 Vorgehensweise

Figur A.51 zeigt eine grobe Skizze des kombinierten Algorithmus zum Protokoll-Lernen. Wegen der Notation vergleiche Abschnitt 4.1.1. In der tatsächlichen Implementierung wird zusätzlich eine minimale Häufigkeit h angegeben, mit der eine PDU-Typen-Sequenz β vorkommen muß, damit eine Ableitung von Attributregeln mittels *LARGE* sinnvoll durchgeführt werden kann. β-Sequenzen, die zu selten in p aufgetreten sind, werden so lange von links verkürzt, bis die Häufigkeitsschwelle h erreicht wird. Dadurch kommen in der resultierenden Protokollmaschine auch Attributregeln über weniger PDUs als die vorgegebene Fenstergröße w vor.

Während *InfSM* direkt und nur einmalig eingesetzt wird, muß *LARGE* für *jeden* Zustand s und *jede* in s endende Folge β von w PDU-Typen getrennt aufgerufen werden, und zwar mit den Attributvektoren aller Fälle aus der Beobachtung p, in denen der Zustand i mit einem Auftreten von β erreicht wird. Die Entscheidung, *ob* ein konkretes Auftreten von β in p tatsächlich in Zustand i endet, kann sehr leicht anhand der aus *InfSM* erhaltenen Abbildung M von Sequenzpositionen auf Zustände getroffen werden. Dies erspart ein zusätzliches Parsieren der Beobachtung mit dem gerade abgeleiteten Automaten, das ohne die Kenntnis von M erforderlich wäre.

4.4.2 Inkrementelle Automatenableitung

Der Aufwand von $O(n^2)$ für den *InfSM*-Algorithmus ist zwar grundsätzlich „gutmütig polynomiell". Für lange Beispielbeobachtungen, die nach dem bisher beschriebenen Konzept in einem Durchlauf von *InfSM* komplett zu bearbeiten wären, wächst jedoch selbst dieser quadratische Aufwand zu schnell. Übrigens gilt $O(n^2)$ nicht nur für den Zeit-, sondern auch für den Speicheraufwand, weil die Relationsmatrix der Ähnlichkeitsrelation gespeichert werden muß. Schon bei 1.000 beobachteten PDUs sind dies eine Million Matrixeinträge, was beim Arbeitsspeicher heutiger Arbeitsplatzrechner noch kein Problem darstellt. Aber bei 10.000 PDUs sind das schon 100 Millionen Einträge. Man sieht, daß *InfSM* für die direkte Verwendung der kompletten Eingabesequenz nicht in jedem Fall geeignet ist. Dies gilt vor allem deshalb, weil der *LARGE*-Algorithmus pro Zustand s und hinführender PDU-Sequenz β nur einen kleinen Bruchteil der Beispielsequenz als Eingabe erhält. Hinreichend große Trainingsmengen, um mit *LARGE* vernünftige Regeln für die PDU-Attribute abzuleiten, erfordern also eine sehr lange Beobachtung p.

Daher wurde ein Verfahren entwickelt, mit dessen Hilfe *InfSM* iterativ auf einzelne Teilabschnitte der beobachteten PDU-Typen-Folge angewendet werden kann. Leider eignet sich *InfSM* aufgrund seiner ganzheitlichen Sicht auf die Eingabesequenz über die Ähnlichkeitsrelation relativ schlecht für eine inkrementelle Arbeitsweise, bei der die bisherige Hypothese anhand neu eintreffender Symbole angepaßt werden müßte. Der verwendete Algorithmus heißt *InfSM** und ist in Figur A.52 dargestellt. Seine Idee besteht darin, die *InfSM*-Prozedur nacheinander auf einzelne Abschnitte der Eingabesequenz α einer festen Fenstergröße N anzuwenden. Um die bei den einzelnen *InfSM*-Läufen erhaltenen Zustände miteinander identifizieren zu können, ohne daß ein topologischer Vergleich der beiden Zustandsgraphen erfolgen muß, verwendet *InfSM** einfach überlappende Abschnitte: Das Fenster wird pro Schritt nur um die halbe Fensterbreite weitergeschoben. Dadurch lassen sich die Zustandsklassen in der ersten Hälfte eines Folgeabschnitts über die Sequenzpositionen mit den Zuständen des vorangegangenen Abschnitts identifizieren. Mit Hilfe der von *InfSM* ermittelten abschnittsinternen Ähnlichkeitsmaße wird die Verbindung zur zweiten Fensterhälfte hergestellt.

Figur A.53 zeigt die verschiedenen Fälle, die auftreten, wenn ein einzelner Funktionswert der Zuordnungsfunktion M von Sequenzpositionen zu Zuständen eines Teilabschnitts in die globale Zuordnungsfunktion G integriert wird. Im Fall a ist die aktuelle Position sowohl in M als auch in der zusammengefaßten Funktion G ein Zustandsrepräsentant, es erfolgt keine Änderung. Im Fall b wird eine Zustandsabbildung aus M in die an dieser Stelle noch nicht geänderte Funktion G übertragen, wie dies insbesondere in der zweiten Hälfte des M-Fensters vorkommt, wenn der entsprechende G-Bereich erstmalig bearbeitet wird. Bei c dagegen sind schon Zuordnungen in G eingetragen, die Position k_2 – in G bisher noch Zustandsrepräsentant – wird aufgrund einer Äquivalenz in M der Position k_1 zugeordnet. In d schließlich ist dargestellt, daß auch zwei bereits bearbeitete Zustandsrepräsentanten – und mit ihnen die zugehörigen Äquivalenzklassen – durch eine Äquivalenz in M zusammengefaßt werden können. Dies bedeutet, daß alle auf k_1 verweisenden G-Einträge später auf k_1 umzusetzen sind. In diesem Fall bleibt in G aber zunächst ein Pfad mit einer Länge größer als eins zurück; dies ist der Grund, warum der Algorithmus mit den beiden *Solange*-Schleifen nach den Repräsentantenknoten in G suchen muß und warum in G abschließend alle Verweise nochmals auf die jeweilige Pfadwurzel umgesetzt werden. In M kommen solche Pfade nicht vor, dort geht jeder Verweis von Anfang an direkt zum Repräsentanten.

Diese Vorgehensweise zur inkrementellen Anwendung von *InfSM* hat den großen Vorteil, daß als Resultat des gesamten Algorithmus wiederum eine Zuordnung von Sequenzpositionen – über die *gesamte* Eingabesequenz – zu den Zuständen des gebildeten Automaten vorliegt. Es ist also zu jeder PDU bereits bekannt, in welchen Zustand sie den Protokollautomaten bringt. Die gewählte Verknüpfungsstrategie ist außerdem eine logische Erweiterung des *InfSM*-Verfahrens: Die bereits festgestellten Äquivalenzen werden unter Forderung nach Transitivität über die gesamte PDU-Sequenz propagiert. Wenn die charakteristischen PDU-Sequenzen mehrmals innerhalb aller Fensterabschnitte vorkommen, liefert *InfSM** also dasselbe Ergebnis wie ein einzelner *InfSM*-Lauf über die gesamte Eingabe.

Bei konstant gesetzter Fenstergröße wächst die Rechenzeit für die *InfSM*-Läufe nur noch linear mit der Eingabelänge n. Allerdings ist der Zeitaufwand der beiden inneren *Solange*-Schleifen im schlimmsten Fall nicht konstant, weil in G ja längere Verweispfade auftreten können. Diese entstehen jedoch nur im Fall d von Figur A.53, wo jeweils zwei bisherige G-Zustände zu einem einzigen vereinigt werden. Die Anzahl solcher Zustandsvereinigungen

im nachhinein ist zumindest durch die Anzahl der pro *InfSM*-Lauf erzeugten Zustände beschränkt. Sie können überhaupt nur dann auftreten, wenn in einem Folgeabschnitt neue Teilsequenzen auftreten. In der Praxis bleibt die mittlere Zahl der Durchläufe der beiden *Solange*-Schleifen also sehr gering. Der Speicheraufwand von *InfSM** ist bei konstanter Fenstergröße im übrigen auf jeden Fall nur $O(n)$.

4.4.3 Sonderbehandlung der Sequenzränder

Der Anfang bzw. das Ende der beobachteten PDU-Sequenz α sind in bezug auf das Ähnlichkeitsmaß *pp* (4.19) „benachteiligt", weil Präfix- bzw. Postfixähnlichkeiten über die Sequenzgrenzen hinaus nicht existieren. Dies kann dazu führen, daß die Zuordnung einiger der ersten und letzten Sequenzpositionen zu einer Äquivalenzklasse nur aus diesem Grund unterbleibt. Als konkrete Auswirkung könnten sich „Sackgassen" im abgeleiteten Automaten für den Beginn und das Ende der Beispielsequenz ergeben.

Als Gegenmaßnahme gegen diesen unerwünschten Effekt werden die ersten und letzten N_0 Sequenzpositionen in α nicht zur Bildung von Zuständen des abgeleiteten Automaten A herangezogen. Für N_0 verwendet *InfSM** den *maximalen* Wert der verwendeten Ähnlichkeitsschwelle N aller *InfSM*-Durchläufe. Zwischen den so festgelegten Randabschnitten sind die Präfix- bzw. Postfixähnlichkeiten nicht mehr in einer Weise eingeschränkt, die sich auf die resultierende Ähnlichkeitsrelation auswirken könnte.

4.4.4 Implementierung

Der Prototyp des Protokollerners wurde in C++ mit Hilfe der Bibliothek LEDA (*library of efficient data types and algorithms*, Bibliothek effizienter Datentypen und Algorithmen) [MNU] und mit einer grafischen Benutzerschnittstelle in Tcl/Tk implementiert. Figur A.54 zeigt ein Beispiel für die Bildschirmanzeige des Lernprogramms.

Dank der LEDA-Bibliothek kann insbesondere der Zustandsgraph des erlernten Automaten sehr effizient und sehr übersichtlich mit Hilfe des LEDA-Datentyps *GRAPH* algorithmisch bearbeitet und schließlich auf Datenträger abgelegt werden. Als Kantenmarkierungen dienen die PDU-Typen, als Knotenmarkierungen treten Klauselllisten von *LARGE* in umgekehrt polnischer Notation auf. Figur A.55 zeigt beispielhaft Ausschnitte aus einem gelernten Protokollautomaten mit Attributregeln. Der obere Teil gehört zur Knotenbeschreibung, der untere zur Kantenbeschreibung. Diese textuelle Notation können auch die LEDA-Werkzeuge zur Visualisierung von Graphen direkt lesen; alle Abbildungen von Zustandsgraphen im vorliegenden Dokument sind auf diese Weise entstanden.

4.5 Analysephase

In der Analysephase muß der gelernte Automat A schließlich zur Prüfung eines neuen, potentiell fehlerbehafteten Kommunikationsverkehrs verwendet werden. Diese Phase kennzeichnet

die eigentliche Anwendung eines lernfähigen Spuranalysesystems und entspricht vom Einsatzkontext her dem in Abschnitt 2.2.1 dargestellten Szenario für den konventionellen Spuranalysator, der ersten Teilaufgabe dieser Arbeit.

Formal spezifiziert wurde die Anwendungsphase bereits in Abschnitt 4.1.1 über das bc-Prädikat (4.2). Hier soll die algorithmische Realisierung erläutert werden.

Weil der Automat A nur die von *InfSM* erzeugten Basiszustände unterscheidet und die restlichen Protokollregeln nicht zustands-, sondern kontextbasiert modelliert, ist das Aufsetzen auf eine laufende Kommunikationsverbindung hier wesentlich einfacher als beim *FollowSM*-Verfahren, das ja auf erweiterten endlichen Automaten basiert. Neben dem ungewissen Anfangszustand können nichtdeterministische Transitionen zu Mehrdeutigkeiten beim aktuellen Basiszustand führen. Aber selbst bei einem DEA existiert keine obere Schranke für Zahl der zu beobachtenden PDUs, bis sich der aktuelle Basiszustand eindeutig bestimmen läßt. Figur A.56 zeigt ein Beispiel für einen deterministischen Protokollautomaten und eine Beobachtung, die sich vom Automaten mehrdeutig parsieren läßt.

Der Analysealgorithmus funktioniert daher so:

1. Beginne mit einer Zustandsmenge $X = S$, die die möglichen aktuellen Basiszustände von A beschreibt.

2. Lies die nächste PDU π der Beobachtung.

3. Bilde X' als Menge aller Zustände $s \in \Sigma$, die

   - von einem Zustand aus X über das Symbol a_π, den PDU-Typ von π, gemäß δ_A erreicht werden können und
   - deren Attributprädikat $\omega_A(s)$ die Attributierung der letzten w PDUs einschließlich π akzeptiert.

4. Melde einen Protokollverstoß, wenn $X' = \emptyset$, und fange wieder bei Schritt 1 an.

5. Setze $X \leftarrow X'$ und gehe zu Schritt 2.

Wenn vom *LARGE*-Algorithmus für einen Zustand s und eine bestimmte PDU-Typen-Folge β, mit der s erreicht werden kann, *kein* ok-Prädikat erzeugt wurde, erfolgt keine Fehlermeldung. Die Analyse begnügt sich in einer solchen Situation mit der Prüfung der Abfolge der PDU-Typen.

Der Aufwand pro PDU liegt bei höchstens $k \cdot |X|$ ok-Klausel-Auswertungen, wenn k die maximale Zahl der Klauseln pro PDU-Kombination in einem Zustand bezeichnet. Da k für den gelernten Automaten A konstant ist und $|X|$ durch die Zustandszahl $|S|$ beschränkt, ist der Zeitaufwand für die Analysephase linear in der Länge der Beobachtung. In der Praxis wird $|X|$ im Mittel sogar nahe bei 1 liegen. Im Vergleich zur Lernphase ist der Einsatz des gebildeten Analysators also sehr effizient.

4.6 Praktische Anwendung

Dieser Abschnitt beschreibt experimentelle Erfahrungen mit dem Prototyp des Protokoll-Lernsystems. Dabei kamen einerseits das Schicht-2-Protokoll der Breitband-ISDN-Signalisierung an der Teilnehmer-Netzwerk-Schnittstelle nach Q.2110 [IT94a], das Protokoll SSCOP (*service specific connection-oriented procedure*, dienstspezifische verbindungsorientierte Prozedur), und andererseits das Internet-Transportprotokoll TCP (*transmission control protocol*, Übertragungssteuerungsprotokoll) [Pos80] zum Einsatz.

4.6.1 Versuche mit SSCOP

Dieser Abschnitt beschreibt die Lernversuche mit dem Protokoll SSCOP.

Da die SSCOP einen selektiven Bestätigungsmechanismus mit einem Sequenznummernraum von 2^{24} Elementen und ein zum Datentransport teilweise asynchrones Nachfrage- und Bestätigungsschema beinhaltet, stellt sie ein besonders gutes Testbeispiel für die arithmetischen Regeln für die PDU-Feldinhalte dar, wie sie von *LARGE* erzeugt werden.

Das größte durchgeführte Experiment verwendete eine PDU-Sequenz aus $n = 1953$ PDUs des Protokolls SSCOP, die aus einer programmähnlichen Beschreibung wesentlicher Protokollprozeduren erzeugt wurde, mit deren Hilfe pro Protokollprozedur wechselnde PDU-Feldinhalte zustande kamen ([Hat98]).

Abstraktionsgrad des Basisautomaten

Tabelle 4.5 gibt einen Überblick über den Zusammenhang zwischen dem Schwellenwert N für die Zustandsäquivalenz und der Größe der gelernten Automaten im „interessanten" Bereich. Zusätzlich sind die Anzahl der Kanten im Zustandsgraphen, also die Zahl der Transitionen des gelernten Automaten, sowie die Anzahl der Zustands-Eingabe-Paare angegeben, für die es jeweils mehrere mögliche Folgezustände gibt. In der Tabelle sind letztere als „nichtdeterministische Situationen" bezeichnet.

Schwelle N	**Zustände** $\|S\|$	**Kanten**	**nichtdet. Situationen**	**Figur**
6	41	56	3	–
5	27	42	2	A.57
4	21	34	2	A.58
3	10	23	1	A.59
2	10	23	1	A.59
1	8	22	0	–

Tabelle 4.5: Verschiedene endliche Automaten für SSCOP in Abhängigkeit von der Ähnlichkeitsschwelle N

In diesen Versuchen wurde *InfSM* in einem einzigen Lauf über die gesamte Eingabesequenz ausgeführt. Bei einem *InfSM*-Fenster von 1000 PDUs und einer daraus resultierenden Anwendung von *InfSM** mit vier überlappenden Teilsequenzen (siehe Abschnitt 4.4.2) ergab sich jedoch stets derselbe Automat. Die Zahlen zu den nichtdeterministischen Situationen in der Tabelle zeigen, daß die in Abschnitt 4.3.5 angestellten theoretischen Überlegungen, warum *InfSM* „bevorzugt" deterministische Transitionen bildet, in der Praxis bestätigt werden.

Die Figuren A.57 bis A.59 zeigen drei verschiedene dieser abgeleiteten endlichen Automaten als Zustandsgraphen. Die SSCOP-PDU-Typen sind mit den Suffixen *_i* (*input*) bzw. *_o* (*output*) als Ein- bzw. Ausgabe-PDUs gekennzeichnet. Figur A.60 bietet eine grafische Darstellung des Zusammenhangs zwischen der Schwelle N und der Automatengröße $|S|$ für einen größeren Wertebereich von N. Die $|S|$-Achse ist hier logarithmisch skaliert, weil die Zustandszahl mit zunehmender Schwelle den sinnvollen Bereich sehr schnell verläßt.

Für die Wahl einer sinnvollen Ähnlichkeitsschwelle N bestand ursprünglich die Idee, daß der Verlauf der Zustandszahl in Abhängigkeit von N einen Rückschluß auf die „richtige" Schwelle zulassen könnte – und damit auf die Zahl der Basiszustände, die der tatsächliche Protokollautomat in der IUT verwendet. Leider hat sich dieser Rückschluß als nicht allgemein möglich erwiesen, wie insbesondere das hier beschriebene Experiment zeigt: Erstens nimmt die Zustandszahl in recht groben Schritten ab, Zwischenwerte sind aufgrund der Definition des Ähnlichkeitsmaßes *pp* (4.19) nicht möglich. Zweitens ist die einzige – allerdings deutliche – Auffälligkeit im Diagramm A.60, nämlich die für $N = 2$ und $N = 3$ identischen resultierenden Automaten mit $|S| = 10$ Zuständen, gerade *kein* Hinweis auf einen besonderen Attraktor innerhalb der Zustandszahlen, der auch aus Anwendungssicht ein Optimum darstellt. Denn ein Blick auf den in Figur A.59 dargestellten Automaten mit 10 Zuständen zeigt, daß hier bereits wesentliche Zustände der Protokollmaschine im Standarddokument zusammengefaßt worden sind: Zustand 0 bildet den Grundzustand der Kommunikation, der offenbar mit dem Zustand 10 (*data transfer ready*, bereit zum Datentransport) aus dem Q.2110-Standard identifiziert werden kann. Aber auch nach einem Abbau der Verbindung durch das Paar *END_i*, *ENDAK_o* (Eingabe einer Abbauanforderung gefolgt von der Ausgabe einer Abbaubestätigung) landet der gelernte Automat über den Zustand 4 wieder im Zustand 0.

Daher muß der Schwellenwert N anhand einer vom Benutzer als Parameter vorgegebenen maximalen Zustandszahl ermittelt werden. „Falsche" Automaten entstehen bei ungeschickter Schwellenwahl nicht, die Spezifikation für das Ergebnis des Protokollernens (vgl. Abschnitt 4.1.1) wird immer erfüllt. Größere Zustandzahlen erhöhen die Genauigkeit der gelernten Automaten, aber auch das Risiko des *overfitting* – des Lernens von Eigenschaften, die die Trainingsmenge, aber nicht das Zielkonzept des Protokolls charakterisieren.

Immerhin lassen sich bereits in dem – von der Zustandsunterscheidung her zu stark abstrahierenden – Automaten aus Figur A.59 die typischen Anfrage-Antwort-Paare des SSCOP-Protokolls ausmachen, unter anderem *POLL* gefolgt von *STAT*, *RS* gefolgt von *RSAK* und *ER* gefolgt von *ERAK*. Zustand 5 bildet einen vom Grundzustand der Kommunikation unterschiedenen Fall einer abgebauten Verbindung, die mit dem Paar *BGN_o*, *BGNAK_i* über den Zustand 6 wieder aufgebaut werden kann. Ein bei Kenntnis des Originalprotokolls „besserer" Automat ist allerdings der in Figur A.58 dargestellte mit 21 Zuständen. Dieser Automat befindet sich nach einem ordentlichen Verbindungsabbau meist in einem anderen als dem Grundzustand 0, nämlich in einem der Zustände 7, 10 oder 15. Daß diese Zustände nicht un-

tereinander zusammengefaßt wurden, liegt letztendlich an der Struktur der Trainingsmenge – es fehlen offenbar zu viele Kombinationen aus den möglichen Prä- und Postfixes des Verbindungsabbaus. Allerdings ist in diesem Automaten auch der Grundzustand, also Zustand 0 aus dem 10er-Automaten, hier bereits auf die Zustände 0 und 2 aufgespalten, wie man z. B. an der *RS_i-RSAK_o*-Sequenz von 0 über 5 nach 2 erkennen kann. Die *SD*-PDU, die von Zustand 2 nach Zustand 0 zurückführt, bewirkt im Protokollstandard keinen Zustandswechsel.

Der Vergleich mit dem noch weniger abstrakten 27er-Automaten in Figur A.57 zeigt, daß dort auch noch der Zustand 1 als Variante des Grundzustands auftritt. Insbesondere die *POLL-STAT*-Paare der SSCOP führen zu Zustandswechseln zwischen 1 und 0.

Zusammenfassend läßt sich sagen, daß der Automat mit 22 Zuständen aus Anwendungssicht den besten Kompromiß darstellt – „zu viele" Zustände sind eher akzeptabel als „zu wenige", weil in der Anwendungsphase durch nichtdeterministische Transitionen ggf. der für den Kommunikationsfortschritt passendere Folgezustand gewählt werden kann. Übermäßige Zusammenlegung von Zuständen führt dagegen immer zum Verlust der Fähigkeit des Analysators, bestimmte Protokollverstöße zu erkennen.

Attributregeln

Um das Lernresultat in bezug auf die Attributregeln zu beleuchten, soll im folgenden der Automat mit 21 Zuständen betrachtet werden. Für ihn wurden für eine maximale Kontextfenstergröße von $w = 3$ aus der Sequenz von 1953 PDUs insgesamt 31 Attributregeln gelernt. Davon beziehen sich 24 Regeln auf eine Dreiersequenz, 5 auf eine Zweiersequenz und 2 Regeln auf Einzel-PDUs. Jede Attributregel mußte auf mindestens 10 Beispielen basieren. Die für den gesamten Lernprozeß – Automat und Attributregeln – benötigte Zeit liegt bei 198 Sekunden auf einem PentiumPro-PC mit 128 MB Hauptspeicher und 200 MHz Takt. Davon entfallen nur etwa 24 Sekunden auf *InfSM*, 174 Sekunden benötigt *LARGE* für das Erlernen der Attributregeln.

Die gelernten Attributregeln können nur beispielhaft untersucht werden. Figur A.61 zeigt ein Beispiel einer Attributregel in der Rohversion, wie sie vom Lernalgorithmus ausgegeben wird.

Um diese zwei Klauseln zu untersuchen, ist zunächst die Bedeutung der v_i zu erklären. Die drei PDUs *SD*, *SD*, *USTAT* liefern die acht Attribute in folgender Weise:

$$\begin{aligned} SD_i \quad & (N(S)_1 = v_1,\ len_1 = v_2) \\ SD_i \quad & (N(S)_2 = v_3,\ len_2 = v_4) \\ USTAT_o \quad & (N(MR)_3 = v_5,\ N(R)_3 = v_6,\ L1_3 = v_7,\ L2_3 = v_8) \end{aligned}$$

Mit diesen Bezeichnungen der PDU-Felder, die aus dem Standard Q.2110 stammen, lassen

sich unter anderem folgende Beziehungen aus den Regeln in Figur A.61 ableiten:

$$1 \leq N(S)_1 \leq 16 \wedge len_1 = 1 \wedge$$
$$N(S)_2 = N(S)_1 + 2 \wedge len_2 = 1 \wedge$$
$$N(MR)_3 = N(S)_2 + 7 \wedge N(R)_3 = N(S)_2 - 1 \wedge 2 \leq L1_3 \leq 17 \wedge L2_3 = N(S)_2$$
$$\vee$$
$$len_1 = 1 \wedge$$
$$N(S)_2 = N(S)_1 + 3 \wedge len_2 = 1 \wedge$$
$$N(MR)_3 = N(S)_2 + 7 \wedge N(R)_3 = N(S)_2 - 1 \wedge L2_3 = N(S)_2 + 1$$

Das Senden von *USTAT* als Antwort auf die zweite *SD*-PDU ist eine Fehlerbehandlung in der SSCOP, hervorgerufen durch einen Sequenzfehler. Eigentlich müßten die Laufnummern $N(S)$ der *SD*-PDUs fortlaufend hochgezählt werden, dies ist bei den beiden hier beschriebenen Situationen aus der Beispielkommunikation nicht der Fall ($N(S)_2 = N(S)_1 + 2$ bzw. $N(S)_2 = N(S)_1 + 3$). Im *USTAT* gibt $N(MR)$ die höchste empfangbare Laufnummer (Empfangsfenster) an, $N(R)$ nennt die letzte übersprungene PDU, $N(S)_2 - 1$ (vgl. [IT94a]).

Natürlich sind die Fenstergröße von 7, die Auslassung genau einer bzw. genau zweier PDUs ebenso wie das in der ersten Klausel kodierte Laufnummernintervall $1 \ldots 16$ Spezialfälle, die aus der spezifischen Trainingsmenge stammen. Dies kann dem Algorithmus jedoch nicht als Fehler angelastet werden, da er ja nicht „wissen" kann, welche dieser Aspekte zum Protokoll gehören und welche nicht.

4.6.2 Versuche mit TCP

Dieser Abschnitt beschreibt die Versuche mit dem Internet-Transportprotokoll TCP.

TCP ist ein verbindungsorientiertes Protokoll mit einem Bestätigungs- und Flußkontrollmechanismus auf Basis eines gleitenden Fensters (*sliding window*). Das TCP-Kommunikationsmodell ist ein virtueller Datenstrom, in dem nicht die Pakete, sondern die einzelnen Oktette numeriert sind. TCP-PDUs werden in der Protokolldefinition [Pos80] als *Segmente* bezeichnet.

Leider gibt es in der Protokolldefinition keine Unterscheidung in verschiedene PDU-Typen, sondern nur ein Feld mit sechs Kennzeichen (*flags*), die angeben, welche Protokollfunktionalitäten in der jeweiligen PDU angesprochen werden. Dies schließt die Festlegung ein, welche PDU-Felder vom Empfänger interpretiert werden sollen. Daher mußte für einen sinnvollen Einsatz des Lernverfahrens eine Einteilung in PDU-Typen vorgenommen werden, die sich an den Kennzeichen der PDUs orientiert. Die sechs Kennzeichen haben die folgenden Bedeutungen:

URG (*urgent pointer field significant*) Es sind „dringende" Daten bis zu einer angegebenen Position im Datenstrom vorhanden. Dies sollte dem Empfänger angezeigt werden.

ACK (*acknowledgement field significant*) Mit dieser PDU werden empfangene Daten bis zu einer angegebenen Position im Datenstrom bestätigt.

EOL (*end of letter*) Erlaubt die Kennzeichnung einer Nachrichtengrenze innerhalb des Datenstroms.

RST (*reset the connection*) Anweisung an den Empfänger, die Verbindung ohne weitere Interaktion zu löschen.

SYN (*synchronize sequence numbers*) Wird beim Verbindungsaufbau verwendet und zeigt den Start der Oktettnumerierung an. Tatsächlich belegt das *SYN*-Bit zum Zwecke der Eindeutigkeit die Sequenznummer vor dem ersten Datenoktett.

FIN (*no more data from sender*) Zeigt an, daß der Sender in dieser Verbindung keine weiteren Daten schicken wird. Das *FIN*-Bit belegt die Sequenznummer nach dem letzten Datenoktett.

Für den Lernversuch wurden nach Maßgabe dieser Kennzeichen die in Tabelle 4.6 aufgezählten PDU-Typen definiert. Die Kennzeichen *RST*, *SYN* und *FIN* bedingen – in dieser Reihenfolge – jeweils eigene PDU-Typen. Zu jedem PDU-Typ ohne *ACK* gibt es ein Pendant mit gesetztem *ACK*. Die für den Protokollablauf unerheblichen Kennzeichen *URG* und *EOL* werden ignoriert.

	Kennzeichen					
PDU-Typ	URG	ACK	EOL	RST	SYN	FIN
RSTA	?	1	?	1	?	?
RST	?	0	?	1	?	?
SYNA	?	1	?	0	1	?
SYN	?	0	?	0	1	?
FINA	?	1	?	0	0	1
FIN	?	0	?	0	0	1
ACK	?	1	?	0	0	0
DATA	?	0	?	0	0	0

Tabelle 4.6: Für den Lernversuch mit TCP definierte PDU-Typen und ihre Beziehung zu den Segment-Kennzeichen von TCP.

Die Beispielspuren für die TCP-Experimente wurden mit Hilfe des Unix-Werkzeugs *tcpdump* [Law98] im laufenden Betrieb eines lokalen Netzes gewonnen. Die PDU-Schnittstelle des Lernprogramms für TCP kann jede mit der Option *-w* von *tcpdump* erzeugte Mitschnittdatei lesen. Da solche Mitschnitte jedoch im allgemeinen mehrere gleichzeitige TCP-Verbindungen enthalten – beziehungsweise die Auswahl der richtigen Quell- und Ziel-Ports zum Herausfiltern einer speziellen Verbindung kaum a priori möglich ist – wurden die Mitschnitte einer Vorverarbeitung mit dem dafür entwickelten Hilfsprogramm *tcp_split* unterzogen: Das Programm erzeugt eine konvertierte Mitschnittdatei, in der die ursprünglich parallelen Verbindungen hintereinander abgelegt sind. Außerdem werden alle Verbindungen entfernt, zu denen innerhalb eines konfigurierbaren Zeitfensters t_{idle} am Beginn oder Ende des ursprünglichen Mitschnitts PDUs aufgetreten sind. Dem liegt die heuristische Annahme zugrunde, daß eine TCP-Verbindung, zu der für eine Zeitspanne t_{idle} keine PDUs beobachtet werden, sich tatsächlich im inaktiven Zustand befindet, also nicht mehr vorhanden ist. Die Verkettung

zweier unterschiedlicher Verbindungen durch *tcp_split* erfolgt, sofern diese Annahme zutrifft, also immer in diesem inaktiven Zustand. Daher ist es sinnvoll, die resultierende Spur insgesamt als Lehrbeispiel für einen Protokollautomaten anzusehen. Darüber hinaus läßt sich eine Minimal- und eine Maximalzahl von PDUs angeben, außerhalb derer eine Verbindung nicht in den konvertierten Mitschnitt übernommen wird. Figur A.62 verdeutlicht die geschilderte Funktionsweise des Werkzeugs *tcp_split* an einem Beispiel.

Hier sollen zwei Lehrbeispiele des Protokolls TCP betrachtet werden, deren Eckdaten in der folgenden Tabelle zusammengefaßt sind. Die drei letzten Spalten beziehen sich auf die Parameter der Vorverarbeitung mit *tcp_split*:

Beispiel	Anzahl PDUs	Verbindungen	Zeitfenster t_{idle} [s]	min. PDUs pro Verbindung	max. PDUs pro Verbindung
A	1625	28	900	40	–
B	11867	385	3600	10	500

Endlicher Basisautomat

Tabelle 4.7 gibt einen Überblick über verschiedene endliche Automaten, die als Lernresultat über die beiden Beispielsequenzen auftreten. Die zweite Sequenz, B, ist mit 11867 PDUs zu lang für einen einzelnen *InfSM*-Lauf, sie wurde daher in 11 *InfSM*-Durchläufen bei einer Fenstergröße von 2000 PDUs mit Hilfe von *InfSM** gelernt. Da sich die vorgegebene Maximalzahl von Zuständen – siehe zweite Spalte der Tabelle – auf die einzelnen Durchläufe bezieht, kann die Zustandszahl des resultierenden Automaten nach der Kombination der einzelnen Zustandsäquivalenzen in beide Richtungen von dieser Vorgabe abweichen. Eine Angabe des Schwellenwertes N als unabhängige Variable, wie in der entsprechenden Tabelle 4.5 für die SSCOP-Experimente, ist hier nicht sinnvoll, da die gewählte Schwelle von Durchlauf zu Durchlauf variiert. Das Diagramm in Figur A.66 stellt den Zusammenhang zwischen Schwelle N und Zustandszahl $|S|$ *für jeden einzelnen Durchlauf* von *InfSM* dar – für die Beispielsequenz B sind also 11 Graphen eingezeichnet.

Beispiel	Soll-Zustände	Zustände $\|S\|$	Kanten	nichtdet. Situationen	Figur
B	8	4	15	3	–
B	10	8	20	2	–
B	11...12	9	22	3	–
B	13...15	8	21	2	A.63
B	16...19	17	34	3	A.64
B	20	22	40	4	–
A	14	12	23	3	A.65

Tabelle 4.7: Verschiedene endliche Automaten für TCP.

Man sieht, daß die gelernten Automaten wiederum nur wenige nichtdeterministische Transitionen aufweisen. Durch das Mehrstufenverfahren *InfSM** ergibt sich eine relativ grobe Abstufung der unterschiedlich abstrakten Automaten. Das Diagramm zeigt sehr große Unterschiede

im Verlauf der Zustandszahl zwischen den verschiedenen Abschnitten der *B*-Sequenz. Besonders dadurch bietet sich auch hier keinerlei Ansatzpunkt für eine automatische Auswahl des „besten" Automaten in bezug auf den Abstraktionsgrad.

Die Figuren A.63 bis A.65 zeigen drei der in Tabelle 4.7 aufgeführten Automaten als Zustandsgraphen. Der Automat in Figur A.63 resultiert aus der langen Beispielsequenz *B*. Er wurde aus Einzelrelationen mit höchstens 13 bis 15 Zuständen zusammengefügt, woraus sich nach der Überlagerung insgesamt 8 Basiszustände ergaben. Zahlreiche Transitionen des Automaten beginnen und enden in Zustand 0, wie man leicht an den übereinandergedruckten Kantenbezeichnern erkennt. Dieser Zustand stellt also den Grundzustand des Kommunikationsablaufs dar und wird während der normalen Datenübertragungsprozeduren nicht verlassen. Leider ist auch der Zustand einer inaktiven Verbindung im Automatenzustand 0 angesiedelt, denn alle *SYN*- und *SYNA*-Kanten beginnen dort. Der in der Figur untere Teil des Automaten modelliert aber gut verschiedene mögliche Abläufe des Verbindungsabbaus mit dem *FIN*-Kennzeichen und seiner Bestätigung. Die in Tabelle 4.7 genannten weiteren Automaten mit 8 bzw. 9 Zuständen unterscheiden sich nur marginal vom abgebildeten.

Figur A.64 zeigt den Automaten mit 17 Zuständen, der sich ergibt, wenn man die Einzelrelationen auf höchstens je 16 Zustände begrenzt. Man erkennt, daß die beiden Zustände 8 und 12 offenbar redundant sind. Dies läßt sich nur damit erklären, daß die *SYN_o-SYNA_i*-Paare im Beispiel *B* so selten bzw. mit teilweise so großem Abstand auftreten, daß trotz 1000 PDUs Überlappung zwischen den Lernabschnitten keine Zusammenfassung der zugrundeliegenden Sequenzpositionen erfolgt ist (vgl. Abschnitt 4.4.2). Gegenüber dem ersten Automaten sind zahlreiche Zustände aufgespalten worden. Bei genauerer Betrachtung wird deutlich, daß die Zustände 0 bis 3 erhalten geblieben sind. Zustand 4 ist allerdings aus dem „Grundzustand" 0 herausgelöst worden, so daß aus 5 und 6 jetzt 6 und 7 geworden ist. Ein Vergleich zwischen dem neuen Sektor links unten im Automaten und der schon bekannten Gruppe der ersten acht Zustände deutet darauf hin, daß bei 17 Zuständen bereits eine übermäßige Verfeinerung eingesetzt hat.

In Figur A.65 ist zum Vergleich ein Automat dargestellt, der aus der mit 1625 wesentlich kürzeren Beispielsequenz *A* abgeleitet wurde. Es fällt auf, daß er kaum Ähnlichkeiten mit den Ergebnissen zur Sequenz *B* aufweist: Sowohl die für feste PDU-Paare zuständigen Zustände 4 und 5 als auch die beiden größeren Schleifen finden keine funktionale Entsprechung in topologisch verwandten Konstrukten der beiden ersten vorgestellten Automaten. Die einzige Gemeinsamkeit besteht offenbar darin, daß ein einziger Zustand den Großteil der Protokollprozeduren alleine übernommen hat. Einige Details – wie die *RST_o-RST_o*-Sequenz hier gegenüber *RST_i-RST_i* in Figur A.64 – deuten darauf hin, daß es signifikante strukturelle Unterschiede in den Trainingsbeispielen gibt, obwohl diese für ein relativ einfaches Protokoll wie TCP bereits recht lang sind.

Attributregeln

Für den vollständigen Lernvorgang einschließlich der Attributregeln wurde der Automat mit 8 Zuständen ausgewählt (siehe Figur A.63). Es wurden Regeln über ein Kontextfenster von höchstens 3 PDUs aus jeweils mindestens 10 Beispielen pro Regel gelernt. Der vollständige Lernvorgang aus 11867 PDUs dauert auf einem PentiumPro-Rechner mit 200 MHz Takt und

128 MB Hauptspeicher 93 min, wovon etwa 6 min auf *InfSM** und 87 min auf *LARGE* entfallen. Gelernt werden 48 Regeln über drei aufeinanderfolgende PDUs, 14 Regeln über zwei aufeinanderfolgende PDUs und 7 Regeln über Einzel-PDUs, insgesamt also 69 Attributregeln.

Als Demonstration einer abgeleiteten Attributregel soll hier die Sequenz *ACK_i*, *FINA_i*, *ACK_o* über die Zustände 0 und 1 zum Zustand 2 dienen. Figur A.67 zeigt wieder die Rohfassung der gelernten Regel.

Zum Verständnis dieser Vorschrift benötigt man die Zuordnung der numerierten Attribute zu den PDU-Feldern:

$$\begin{array}{ll} ACK_i & (con_1 = v_1,\ SEQ_1 = v_2,\ WND_1 = v_3,\ flags_1 = v_4,\ ACK_1 = v_5) \\ FINA_i & (con_2 = v_6,\ SEQ_2 = v_7,\ WND_2 = v_8,\ flags_2 = v_9,\ ACK_2 = v_{10}) \\ ACK_o & (con_3 = v_{11},\ SEQ_3 = v_{12},\ WND_3 = v_{13},\ flags_3 = v_{14},\ ACK_3 = v_{15}) \end{array}$$

Während *SEQ*, *WND* und *ACK* PDU-Felder gemäß der TCP-Definition [Pos80] sind, sind die beiden übrigen Feldarten an das Lernproblem angepaßte Repräsentationen:

con Bezeichnet die zugehörige Verbindung (*connection*) und wird aus Quell- und Ziel-Port-Nummer von TCP gebildet. Mit Hilfe dieses Feldes lassen sich Kontextregeln über ein Verbindungsende hinaus erkennen.

flags Bitsignifikanter Wert, der sich aus denjenigen Kennzeichen aus der TCP-PDU zusammensetzt, die nicht aus dem zugewiesenen PDU-Typ hervorgehen (vgl. Tabelle 4.6).

Auf Basis dieser Feldbezeichner kann die Regel aus Figur A.67 umformuliert werden zu:

$$\begin{array}{ll} & con_1 = con_2 = con_3 \\ \wedge & ACK_3 = SEQ_1 + 1 = SEQ_2 + 1 \\ \wedge & WND_1 = WND_2 = WND_3 = 8760 \\ \wedge & flags_1 = flags_2 = flags_3 = 0 \\ \wedge & SEQ_3 = ACK_2 = ACK_1 \end{array}$$

Diese Beziehungen sagen aus, daß die drei PDUs zur selben TCP-Verbindung gehören und daß keine weiteren Kennzeichen (außer *ACK* und *FIN*) verwendet werden. Insbesondere der typische Mechanismus, daß das *FIN*-Bit eine eigene Sequenznummer trägt, die mit ACK_3 bestätigt werden muß, wird genau abgebildet. Beim festen Wert der Empfangsfenster-Felder *WND* wird aber wieder deutlich, daß selbst eine so große und unter völlig realistischen Bedingungen aufgenommene Trainingsmenge nicht vor implementierungsspezifischen Besonderheiten schützt.

Kapitel 5

Bewertung und Ausblick

Dieses Kapitel bewertet den praktischen Nutzen der beiden in dieser Arbeit entwickelten Verfahren und untersucht, wie sinnvolle Erweiterungen und Ergänzungen dieser Ansätze aussehen könnten.

5.1 Vergleichende Bewertung

Zunächst soll die praktische Brauchbarkeit der entwickelten Verfahren, Spuranalyse mit *FollowSM* und Protokollernen mit *InfSM* und *LARGE*, sowohl pro Verfahren als auch vergleichend bewertet werden. Die Kriterien zur Bewertung gliedern sich in die *Vollständigkeit* der Fehlererkennung, die *Korrektheit* von Meldungen über Fehler, die Transparenz der Analyseergebnisse für den Benutzer in Abhängigkeit von dessen Sachkenntnis, den Aufwand für die Erzeugung eines Analysators für ein konkretes Protokoll sowie den Aufwand der eigentlichen Analyse.

5.1.1 Vollständigkeit

Mit *Vollständigkeit* ist die Vollständigkeit der Fehlerdetektion gemeint, also die Frage, ob *alle* denkbaren Arten von Fehlverhalten relativ zum zugrundeliegenden Kommunikationsprotokoll vom Analyseverfahren erkannt werden. Um die Dualität zur Frage der Korrektheit zu verdeutlichen, entspricht das der Frage, ob aus dem Auftreten eines Protokollverstoßes grundsätzlich eine Fehlermeldung folgt.

Beim *FollowSM*-Verfahren wird, sobald die Synchronisationsphase beendet ist und die Zustandsbeschreibungen keine unsicherheitsbehafteten Einträge mehr enthalten, ein gewisses Maß an Vollständigkeit erreicht: Alle *im OTEFSM-Modell enthaltenen* Protokollregeln werden dann vollständig überwacht. Allerdings gibt es Fälle, in denen man auf eine derartige vollständige Modellierung verzichten muß:

- Die *Unabhängigkeitsprämisse* (vgl. Abschnitt 3.3.4) erlaubt keine expliziten Manipulationen eines Systemzustands, der mehrere Einzelautomaten umfaßt. Protokolle, die

darauf angewiesen sind, können nur unter Verzicht auf Vollständigkeit in den entsprechenden Situationen modelliert werden.

- Eine Abbildung sehr komplexer Datenstrukturen im Protokollzustand in die unsicherheitsbehaftete Zustandsrepräsentation in *FollowSM* kann dazu führen, daß der Suchaufwand im Zustandsraum zu groß wird oder sogar Endlosschleifen im unsicherheitsbehafteten Zustandsraum entstehen. Letzteres geschieht, wenn die Voraussetzung für das Schalten einer Transition auch im erzeugten Folgezustand erhalten bleibt. In solchen Fällen muß der günstigste Fall im Hinblick auf korrektes Protokollverhalten angenommen, mithin auf Vollständigkeit verzichtet werden.

- Aus Sicht des Benutzers verletzt schon die – formal unvermeidliche – unvollständige Fehlererkennung in der Synchronisationsphase (vgl. Theorem 4 in Abschnitt 3.2.2) das Vollständigkeitsgebot.

Trotzdem weist das *FollowSM*-Verfahren eine sehr hohe Fehlererkennungsquote auf, weil es den Datenanteil des Protokollzustands, die Datenfelder der PDUs und Zeitbedingungen überprüft und Ursache und Wirkung von Protokollereignissen präzise modelliert sind. Die experimentelle Bestätigung für die gute Fehlererkennungsfähigkeit wurde in Abschnitt 3.5.3 im Zusammenhang mit den Versuchen bei der GMD erbracht.

Für die Spuranalyse in der Anwendungsphase eines erlernten Protokollautomaten kann dagegen unter keinen Umständen Vollständigkeit zugesichert werden. Dies liegt daran, daß sowohl der *LARGE*-Algorithmus beim Lernen der Attributprädikate als auch der *InfSM*-Algorithmus bei der Erzeugung des PDU-Typen-Automaten zwangsläufig über die Lehrbeispiele hinaus generalisieren müssen, weil die Trainingsmenge nur eine kleine Stichprobe des korrekten Verhaltens darstellt. Außerdem hängt das Fehlererkennungsvermögen eines gelernten Analyseautomaten natürlich sehr stark von der Qualität des verwendeten Trainingsverkehrs ab, welche sich wiederum ohne a priori vorhandenes Protokollwissen kaum abschätzen läßt. Obwohl sich diese Unzulänglichkeiten aus der Natur der Sache ergeben, lassen sie sich insbesondere im Hinblick auf ein kommerzielles Produkt dem Benutzer eines Spuranalysators nur schwer vermitteln.

5.1.2 Korrektheit

Mit *Korrektheit* ist die Korrektheit der Fehlerdetektion gemeint, d. h. die Frage, ob es tatsächlich nur bei Vorliegen eines Protokollverstoßes zu einer Fehlermeldung des Analysators kommen kann. Dies entspricht der Frage, ob aus dem Erhalt einer Fehlermeldung grundsätzlich das Vorliegen eines Protokollverstoßes folgt.

Bei der Spezifikation der unsicherheitsbehafteten Spuranalyse in Abschnitt 3.2.2 wurde der Korrektheit unbedingter Vorrang gegenüber der Vollständigkeit eingeräumt. Dies ist sinnvoll, weil die hier betrachteten Spuranalysatoren in einem Kontext zum Einsatz kommen, in dem sonst *keinerlei* Fehler gefunden werden. Unvermeidliche Unzulänglichkeiten eines Analysators sollten sich eher in einer nur teilweisen Lösung der Erkennungsaufgabe äußern als den Benutzer mit einer Flut von unzutreffenden Fehlermeldungen zu konfrontieren. Dies gilt

ganz besonders dann, wenn der Benutzer selbst gar nicht über die erforderlichen Kenntnisse verfügt, um im Einzelfall zutreffende von unzutreffenden Fehlermeldungen zu unterscheiden.

Beim *FollowSM*-Verfahren kann die Korrektheit aller Fehlermeldungen garantiert werden, sofern bei der OTEFSM-Modellierung keine Entwurfsfehler unterlaufen sind (vgl. Theorem 3 in Abschnitt 3.2.2). Diese Aussage relativiert sich nur insoweit, daß der *gemeldete* Fehler nicht immer mit der tatsächlichen internen *Fehlfunktion* der IUT übereinstimmen muß, weil der eigentlich zutreffende Pfad im Zustandsraum durch die Fehlfunktion früher verworfen worden sein kann als ein oder mehrere zunächst möglich erscheinende Alternativpfade (vgl. Abschnitt 3.3.6).

Beim Resultat des Lernverfahrens kann auch die Korrektheit der Fehlermeldungen nicht garantiert werden, da man nie ausschließen kann, daß irgendeine korrekte Protokollprozedur während der Lernphase nicht aufgetreten ist und dann, in der Analysephase beobachtet, zu einer unzutreffenden Fehlermeldung führt.

5.1.3 Transparenz der Diagnosen

Dieses Kriterium zielt darauf ab, ob und inwiefern die vom Analysator erzeugten Diagnose- und Fehlermeldungen für den Benutzer hilfreich sind, um die zugrundeliegenden Fehlfunktionen zu beseitigen. Das ist ja das eigentliche Ziel beim Einsatz eines Spuranalysators in dem in dieser Arbeit angenommenen Kontext.

Das *FollowSM*-Verfahren erreicht eine gute Aussagekraft der Diagnosemeldungen. Besonders die in Abschnitt 3.3.6 beschriebenen Ergebnisberichte auf höherer Abstraktionsebene, darunter die Parameterschätzung und die Fehlerklassifikation mit -statistik, sind aus Benutzersicht wertvoll und vermeiden dabei eine Überfrachtung mit Detailinformationen.

Die hohe Aussagekraft erreicht *FollowSM* dadurch, daß es auf Begriffe und Bedeutungen aus der originalen Protokollspezifikation zurückgreifen kann, um die erkannten Fehlersituationen und Fehlfunktionen zu charakterisieren. Dies bedeutet natürlich auch, daß ein Benutzer über Kenntnisse des Zielprotokolls verfügen muß, um die von *FollowSM* erzeugten Ergebnisberichte vollständig nutzen zu können. Die Situation zeigt Parallelen mit den Fehlermeldungen eines Übersetzers für eine Programmiersprache: Wer die jeweilige Programmiersprache nicht kennt oder sogar überhaupt nicht programmieren kann, wird mit den Fehlermeldungen eines Übersetzers auch nichts anzufangen wissen. Aber ein Benutzer des Übersetzers, der die Programmiersprache beherrscht, ist darauf angewiesen, daß die Fehlermeldungen sehr genau auf Details der Fehlersituation und den Zusammenhang zur Sprachdefinition eingehen. Ebenso erlauben die Fehlerbeschreibungen von *FollowSM*, die eine Fehlersituation ja durch die letzte passende Zustandsbeschreibung und den Grund für das Scheitern ihrer Fortschreibung charakterisieren, vor allem einem Protokollexperten eine sehr effiziente und tiefgehende Analyse der vorliegenden Fehlfunktion in der IUT.

Die Analysephase beim Lernverfahren ist in der Aussagekraft der Meldungen deutlich unterlegen. Zwar können auch hier die auftretenden Fehlerereignisse durch das letzte Teil des durchlaufenen Pfades im Automaten der PDU-Typen dokumentiert werden und dadurch, ob der nächste PDU-Typ oder die Attributierung der nächsten PDU für das Scheitern der Zustandsverfolgung verantwortlich war. Weil der Analyseautomat aber erlernt ist und keinerlei

Informationen über die Semantik seiner Zustände und Attributprädikate zur Verfügung stehen, ist der Nutzen solcher Angaben im Hinblick auf eine Fehlerbeseitigung äußerst begrenzt. Außerdem müssen die Effekte möglicher Nachrichtenkreuzungen (vgl. Abschnitt 2.2.1) durch alternative Abläufe im Automaten nachgebildet werden, was diesen weniger abstrakt und logisch macht als den Automaten, der den Protokollentwurf beschreibt. Ob Fehler nur der IUT oder auch der Partnerinstanz erkannt werden, hängt leider ausschließlich vom Charakter des Beispielverkehrs ab, nämlich von der An- oder Abwesenheit von Protokollverstößen der Partnerinstanz darin, wie in Abschnitt 2.2.2 geschildert. Am ehesten eignen sich die Ausgaben des gelernten Analysators für einen Vergleich verschiedener Kommunikationsstrecken oder -systeme, wie ja das Verfahren im Grunde immer auf einen Vergleich zwischen der zum Training verwendeten Beispielkommunikation mit den später beobachteten Kommunikationsabläufen hinausläuft. Allerdings wäre es keineswegs sachgerecht, einem Verfahren, daß gerade ohne eine Protokollspezifikation für das Zielprotokoll auskommen soll (vgl. Einleitung), die mangelnde Fähigkeit vorzuwerfen, auf eine Protokollspezifikation bezogene Diagnosen zu produzieren. Letztlich setzt die beschriebene Relativität der Analyseergebnisse des Lernverfahrens die Fähigkeit des Benutzers voraus, diese Ergebnisse angemessen zu interpretieren. Damit ist auch in diesem Fall einem Protokollexperten durch das Werkzeug mehr gedient als einem Anwender ohne spezielle Sachkenntnis.

5.1.4 Anpassung an das Zielprotokoll

Die Anpassung des Verfahrens an ein neues Zielprotokoll, also die Generierung eines protokollspezifischen Analysators, kommt beim *FollowSM*-Verfahren nicht ohne einen gewissen Anteil an Entwurfsarbeit aus; selbst dann nicht, wenn eine SDL-Spezifikation zur automatischen Ableitung des Gerüsts der OTEFSM-Spezifikation für *FollowSM* verwendet wird (vgl. Abschnitt 3.4.2). Allerdings kann die Verwendung der SDL-Spezifikation die Entwicklungszeit drastisch verkürzen. Während die Anpassung an ein komplexes, reales Protokoll bei verfügbarer SDL-Spezifikation, aber manueller Erstellung der OTEFSM-Spezifikation in der Größenordnung eines Personenmonats dauert, läßt sich diese Zeit unter Verwendung des beschriebenen SDL-Übersetzers auf etwa ein Viertel reduzieren. Dieser Anpassungsaufwand dürfte für zahlreiche Anwendungen akzeptabel sein.

Die wesentliche Stärke des Lernverfahren ist, daß es den Anpassungsaufwand pro Zielprotokoll minimiert, insbesondere den Anteil manueller Arbeit dabei. Diese beschränkt sich auf die Erstellung eines einfachen Protokoll-Dekoders, weil die Herleitung der Attribute aus dem Nachrichtenstrom ja aus der Lernaufgabe ausgeklammert worden ist (vgl. Abschnitt 4.1.1). Ein solcher läßt sich in vielen Fällen innerhalb eines einzelnen Arbeitstages erstellen. Anschließend ist ein Beispielverkehr aus beispielsweise 100.000 PDUs aufzunehmen und auszuwerten. Die Zeit für die Aufnahme hängt natürlich vom Kommunikationsaufkommen auf dem Beispielkanal ab. Die reine Rechenzeit für die Auswertung wird nach den mit dem Prototyp erzielten Ergebnissen aus Abschnitt 4.6 dann in der Größenordnung eines Tages liegen. Besonders hervorzuheben ist, daß sowohl die Aufnahme als auch die Auswertung des Beispielverkehrs *ohne jedes menschliche Zutun* erfolgen können und daher nur sehr geringe Kosten verursachen.

5.1.5 Aufwand der Analyse

Der Rechenzeitaufwand für den Analysealgorithmus ist in beiden Fällen linear in der Beobachtungsdauer und der Speicheraufwand konstant (siehe Abschnitte 3.3.3 und 4.5), was beide Verfahren für eine Echtzeitanalyse synchron zur Beobachtung qualifiziert. Welche Bandbreiten konkret als Obergrenzen für einen solchen „Online"-Betrieb anzusetzen sind, hängt im Einzelfall sowohl von der verwendeten Hardware des Analysators als auch von der Komplexität des Zielprotokolls ab. Der in Abschnitt 3.5.3 erhaltene Wert von 200 bis 400 analysierten PDUs pro Sekunde auf einem Arbeitsplatzrechner von recht geringer Rechenleistung hat gezeigt, daß „Online"-Analysen in bestimmten Anwendungsfällen möglich sind. Im Vergleich ist beim Lernverfahren eher mit niedrigerem Zeitbedarf zu rechnen, weil die Suche im Zustandsraum hier stets linear fortschreitet.

Trotzdem darf man nicht übersehen, daß beim Stand der Technik in der Breitbandkommunikation, wo viele Protokolle bis mindestens in die OSI-Schicht 2 in Hardware implementiert werden müssen, um mit der Datenrate mithalten zu können, Spuranalysatoren in der Verarbeitungsgeschwindigkeit *immer* hinter den schnellsten Protokollimplementierungen zurückbleiben, weil sie komplizierter sind und ihre Implementierung in Hardware wirtschaftlich nicht lohnt.

5.1.6 Resümee

Tabelle 5.1 stellt den Vergleich der untersuchten Verfahren anhand der genannten Merkmale zusammengefaßt dar. Als Resümee läßt sich aus Sicht des Autors folgendes feststellen:

1. Das *FollowSM*-Verfahren zeigt zweifelsfrei einen hohen Nutzeffekt im praktischen Einsatz, sofern die Anwendung die Kosten der Protokollanpassung rechtfertigt. Es eignet sich allerdings besser als hochspezialisiertes Werkzeug für Spezialisten denn als universelles „Troubleshooting-Tool" für die Endanwender der Kommunikationssysteme.

2. Das Lernverfahren ist vom praktischen Nutzwert demgegenüber stark eingeschränkt. Der gelernte Analysator kann aber immerhin verwendet werden, um einen ersten Eindruck von der Korrektheit einer Protokollimplementierung zu erhalten. Angesichts seiner sehr niedrigen Rentabilitätsschwelle infolge der minimalen Anpassungskosten und weil keine Informationen über das korrekte Protokollverhalten benötigt werden, sind sinnvolle Anwendungsgebiete des Lernansatzes durchaus vorhanden. Der Lernansatz liefert insbesondere eine sehr stark komprimierte Charakterisierung des jeweiligen Lehrbeispiels. Der gelernte Automat beschreibt nämlich *sämtliche* Situationen, die im Lehrbeispiel aufgetreten sind. Damit stellt das Erlernen eines Protokollautomaten eine sehr sinnvolle Maßnahme dar, um einen ersten Überblick über das Verhalten eines zu analysierenden Zielsystems zu erhalten..

5.2 Ansätze für Weiterentwicklungen

In diesem Abschnitt sollen Möglichkeiten erörtert werden, wie die entwickelten Verfahren weiterentwickelt werden könnten, um ihren Einsatz zu vereinfachen und ihren Nutzen zu

Kriterium	***FollowSM*-Verfahren**	**Lernverfahren *InfSM/LARGE***
Vollständigkeit	Erkennt nach der Synchronisationsphase *alle* Verstöße gegen *modellierte* Regeln.	Erkennt nicht alle Fehler, abhängig von der Trainingskommunikation.
Korrektheit	Alle Fehlermeldungen sind zutreffend.	Fehlermeldungen können auch aus Lücken in der Trainingskommunikation resultieren.
Aussagekraft der Diagnose	Äußerst hilfreich und arbeitssparend für Protokollexperten. Bedingt hilfreich für Endanwender.	Grobe Einstufung für Protokollexperten. Für Endanwender kaum zu empfehlen.
Anpassung an neue Protokolle	Größenordnung eine Personenwoche Entwicklungsarbeit bei Verwendung einer SDL-Spezifikation.	Größenordnung ein Personentag Entwicklungsarbeit zuzüglich einige Stunden bis wenige Tage Rechenzeit für Aufnahme der Trainingskommunikation und Lernprozeß.
Aufwand bei der Analyse	Gering. Zeitbedarf linear in der Beobachtungsdauer, Speicherbedarf konstant. In vielen Fällen „Online"-Analyse möglich.	Gering. Zeitbedarf linear in der Beobachtungsdauer, Speicherbedarf konstant. Konstanten kleiner als bei *FollowSM*. In vielen Fällen „Online"-Analyse möglich.

Tabelle 5.1: Gegenüberstellung der in dieser Arbeit untersuchten Verfahrenstypen anhand der wesentlichen Bewertungskriterien.

erhöhen. Die entscheidenden Ansatzpunkte für Verbesserungen sind:

1. Die mangelnde Eignung beider Verfahren für Anwender ohne Expertenwissen über das Zielprotokoll.
2. Der Bedarf nach manueller Unterstützung bei der Erstellung eines protokollspezifischen Analysators nach dem *FollowSM*-Verfahren.

Folgende Ideen für Verbesserungen werden in den nächsten Abschnitten betrachtet: Die Verwendung von Hybridverfahren, die Nutzung eines Expertensystems als Interpretationsschicht und eine bedingungslose SDL-Basierung der Spuranalyse ohne manuelle Nachbearbeitung.

5.2.1 Hybridverfahren

Mit *Hybridverfahren* ist gemeint, daß mehrere Analyseverfahren *gleichzeitig* verwendet werden, um die Stärken der Einzelverfahren zu kombinieren und ihre Schwächen auszugleichen. Diese Idee stammt aus der Anfangsphase der Entwicklung der einzelnen Analyseverfahren.

Die Nutzung eines Hybridverfahrens ist dann theoretisch sinnvoll, wenn unterschiedliche Verfahren vorliegen, die zueinander orthogonale Vorteile in einem oder mehreren bestimmten Qualitätskriterien aufweisen, um insgesamt eine Verbesserung in genau diesem bzw. diesen Qualitätkriterien zu erreichen. Dazu ein Beispiel: Wenn ein Analyseverfahren vor allem Verstöße gegen Zeitbedingungen findet und ein anderes vor allem falsch kodierte PDUs, so könnte eine Kombination beider Verfahren eine wirkliche Verbesserung hinsichtlich des Vollständigkeitskriteriums bringen.

Die beiden hier entwickelten Ansätze sind aber nicht in der Ausprägung einzelner Qualitätskriterien zueinander orthogonal, sondern in den Voraussetzungen ihrer Anwendung: Das Lernverfahren kann in Fällen benutzt werden, in denen nötige Informationen für die Anwendung von *FollowSM* fehlen, dessen Einsatz also ausgeschlossen ist. *FollowSM* gleicht dies durch *insgesamt bessere* Analysequalität in allen qualitätsbezogenen Kriterien aus. Eine Kombination macht unter diesen Voraussetzungen keinerlei Sinn. Denn immer *wenn* ein *FollowSM*-Analysator zur Verfügung gestellt werden kann, *dann* ist der Einsatz von *InfSM* und *LARGE* vollkommen überflüssig. In Bezug auf den Zeitaufwand der Analyse selbst kann ein Hybridverfahren natürlich niemals eine Verbesserung gegenüber einem der Einzelverfahren bedeuten.

Hybridverfahren sind also prinzipiell sinnvoll, aber nicht im Zusammenhang mit den beiden hier untersuchten Ansätzen, die im Raum der möglichen Verfahren diagonal gegenüberliegende Extrempunkte in den Dimensionen Anpassungsaufwand und Ergebnisqualität einnehmen.

5.2.2 Expertensystem als Interpretationsschicht

Eine mögliche Abhilfe für die Lücke zwischen der gegebenen Diagnose eines Analysators und geeigneten Maßnahmen zur Problembeseitigung, also eine Weiterentwicklung im Hinblick auf den den obigen Ansatzpunkt 1, wäre die Verwendung eines Expertensystems als zusätzliche Interpretationsschicht.

Dies bedeutet, daß eines der Analyseverfahren weitgehend unverändert eingesetzt werden könnte, um *Merkmale* der beobachteten Kommunikation hinsichtlich ihrer Protokollkonformität zu gewinnen. Diese Merkmale sind die Diagnosen der bekannten Verfahren. Die zusätzliche Schicht in Form eines Expertensystems würde als Vermittler zwischen dem Analyseverfahren und einem Benutzer auftreten, der kein Protokollexperte ist. Diese Schicht könnte in der Gesamtheit der Korrektheits- und Unkorrektheitsmerkmale ein typisches Fehlerbild isolieren und z. B. vorbereitete Hinweise geben, wie sich dieses beseitigen läßt.

Obwohl dieser Ansatz sehr vielversprechend ist, stellt er jedoch hohe Anforderungen an seine Realisierung:

- Man benötigt eine große Falldatenbasis, um die Regeln für das Expertensystem aufzustellen. Diese Datenbasis muß nicht nur ein vollständiges Bild der Fehlerdiagnosen eines Analyseverfahrens für sehr viele verschiedene Fehlersituationen enthalten, sondern zu jeder dieser Fehlersituationen auch noch eine präzise Beschreibung des Fehlerbildes und möglicher Gegenmaßnahmen.

- Die im vorangegangenen Punkt beschriebene Falldatenbasis ist im allgemeinen nur für eine ganz bestimmte Konfiguration eines Kommunikationssystems gültig. Besonders die Gegenmaßnahmen müssen direkt Bezug nehmen auf die konkrete Systeminstallation. Ein Beispiel, diesen Sachverhalt zu verdeutlichen, ist eine empfohlene Gegenmaßnahme wie: „Gehen Sie in Raum 4711 und stecken Sie den dritten Stecker von links unter dem Fenster wieder in die kleine runde Anschlußdose."

- Die Zuordnung von Gegenmaßnahmen zu Fehlerbildern ist grundsätzlich mit ähnlichen Schwierigkeiten behaftet wie das Lernen eines Protokolls beim Lernansatz: Die Aufgabe des Expertensystems kann ebenfalls als generalisierendes Lernproblem aufgefaßt werden. Daher läßt sich niemals garantieren, daß die vorgeschlagenen Gegenmaßnahmen korrekt und angemessen sind.

5.2.3 Spuranalyse auf direkter Grundlage von SDL

Der Ansatzpunkt 2, die Notwendigkeit manueller Hilfe bei der Protokollanpassung, läßt sich am ehesten angehen, indem das interne Protokollmodell des Analysators – eine OTEFSM-Spezifikation bei *FollowSM* – direkt auf den Automaten gemäß der SDL-Semantik umgestellt wird. Dadurch werden Unstimmigkeiten bei der Modellierungsmächtigkeit und Schwierigkeiten bei der Konvertierung mit einem Schlag vermieden.

Allerdings hat die in Abschnitt 3.1.2 nachgewiesene Berechenbarkeitsproblematik nichts mit dem *FollowSM*-Verfahren zu tun. Sie würde auch für den Versuch gelten, eine SDL-Spezifikation anhand einer Beobachtung auszuführen. Außerdem gäbe es in einer SDL-Spezifikation ständig Transitionen, die ausführbar, aber nicht beobachtbar sind, weil sie von Ereignissen an der Dienstschnittstelle oder rein internen Signalen ausgelöst werden.

Eine mögliche Lösung, die diese unbeobachtbaren Ereignisse berücksichtigt, ohne eine unendliche Suche zu provozieren, besteht darin, alle von unbeobachtbaren Transitionen potentiell betroffenen Zustandsvariablen oder Komponenten von solchen ständig als „unsicher" zu kennzeichnen. Gleiches müßte für Zustandsvariablen gelten, die Gegenstand komplizierter

Verzweigungsbedingungen werden. Damit erhebt sich aber sofort das Problem, daß die Zustandsbeschreibung möglicherweise nie zu sicheren Variablenwerten konvergiert; und wenn sie einmal konvergiert, läßt sich nicht – wie in Theorem 4 für *FollowSM* – gewährleisten, daß sie sicher bleibt.

Obwohl diese Überlegungen nicht darauf hindeuten, daß das Mitverfolgen einer SDL-Spezifikation für jede Spezifikation möglich bzw. nützlich ist, erscheint diese Weiterentwicklung aus Sicht des Autors als die interessanteste. Es besteht die Hoffnung, daß zahlreiche SDL-Spezifikationen „gutmütig genug" sind, um von dem hier skizzierten Ansatz profitieren zu können. Schlimmstenfalls verspricht er wenigstens eine weitere Reduktion des manuellen Anpassungsaufwands, für die man aber gegenüber *FollowSM* Abstriche bei der Ergebnisqualität riskiert.

Kapitel 6

Zusammenfassung

In dieser Arbeit wurden zwei neue Verfahren der Spuranalyse entwickelt, mit deren Hilfe sich die Abwicklung eines Kommunikationsprotokolls durch passive Beobachtung der ausgetauschten Nachrichten automatisch auf Verstöße gegen die Protokollspezifikation analysieren läßt. Beobachtet wird nur der Nachrichtenaustausch über das Kommunikationsmedium, nicht die Ereignisse an den Dienstschnittstellen der beobachteten Instanz. Es wurde gezeigt, daß hier ein theoretisch nicht berechenbares Problem vorliegt. Die beiden entwickelten Verfahren unterscheiden sich wesentlich in der Art, wie die Protokollspezifikation als Referenz der Spuranalyse gewonnen wird: Das *FollowSM*-Verfahren basiert auf einer an die Anforderungen der Spuranalyse angepaßten formalen Spezifikation, die entweder manuell entwickelt oder zu großen Teilen aus einer Protokollspezifikation in der Standard-Spezifikationssprache SDL automatisch abgeleitet wird. Das zweite Verfahren aus den Teilalgorithmen *InfSM* und *LARGE* dagegen setzt eine protokollkonforme Beispielkommunikation voraus, aus der die Protokollregeln in einem in zwei Phasen aufgeteilten Lernvorgang gewonnen werden. Aufgrund der vorliegenden Lernsituation kann diese Lernaufgabe nur heuristisch gelöst werden.

Das *FollowSM*-Verfahren schließt den Datenanteil eines Protokolls und spezifizierte Zeitbedingungen in die Analyse ein. Es kann jederzeit mitten in einer laufenden Kommunikation gestartet werden. Es wurde gezeigt, daß das *FollowSM*-Verfahren relativ zur Referenzspezifikation *korrekt* und nach einer Synchronisationsphase auch *vollständig* ist. Es bietet dem Anwender sehr hilfreiche Informationen über den beobachteten Kommunikationsablauf auf verschiedensten Abstraktionsebenen und kann für einen Protokollexperten die Sucharbeit nach Fehlerquellen um Größenordnungen reduzieren. Der Zeitaufwand der Analyse ist linear in der Beobachtungsdauer, wodurch „Online"-Analysen in Echtzeit in vielen Fällen möglich sind.

Das zweistufige Lernverfahren leitet mit dem neuen Algorithmus *InfSM* im ersten Schritt einen endlichen Automaten ab, der die Sprache der protokollkonformen Folgen von PDU-Typen definiert. Als wichtiges theoretisches Resultat dieser Arbeit konnte gezeigt und argumentiert werden, daß der *InfSM*-Algorithmus bei nur $O(n^2)$ Zeitaufwand in der Länge der Beispielsequenz *alle optimalen* endlichen Automaten liefert, die die Beispielsequenz akzeptieren. Die gelernten Automaten enthalten zudem *überwiegend deterministische* Transitionen.

Der ebenfalls neue Lernalgorithmus *LARGE* dient dazu, Regeln für die Datenfeldinhalte einer Anzahl aufeinanderfolgender PDUs zu lernen. *LARGE* lernt arithmetische Klassifikationsregeln aus unklassifizierten ausschließlich positiven Beispielen, bildet dazu arithmetische Terme

aus den Attributen der Beispiele und findet selbst eine Klasseneinteilung.

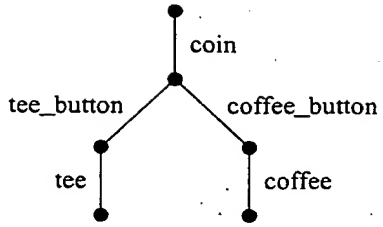
In der Anwendungsphase werden die von *InfSM* und *LARGE* gelernten Regeln wiederum zur Spuranalyse eines Kommunikationsablaufs eingesetzt. Durch den Lernprozeß ist der Arbeitsaufwand zur Erzeugung eines protokollspezifischen Analysators minimal. Die Qualität des Analyseergebnisses bleibt jedoch – aufgrund des immer stichprobenartigen Lehrbeispiels unvermeidlich – klar hinter dem von *FollowSM* zurück. Weder Vollständigkeit noch Korrektheit der Fehlererkennung lassen sich garantieren, weshalb vom Anwender des Analysators Einblick in die Konsequenzen des Lernparadigmas zu fordern ist. Trotzdem gibt es Fälle, in denen sich der Lernansatz sinnvoll praktisch einsetzen läßt.

Es wurden schließlich mögliche Weiterentwicklungen der verfolgten Ansätze untersucht, um auch ohne Expertenwissen zu fehlerbehebenden Maßnahmen zu gelangen und um auf manuelle Entwicklungsarbeit bei der Anpassung an ein Zielprotokoll gänzlich verzichten zu können. Den letzteren Punkt könnte ein *FollowSM*-ähnliches System, das aber komplett auf SDL basiert, leisten. Trotz einiger Einschränkungen bildet dies die interessanteste Entwicklungsperspektive zum behandelten Thema.

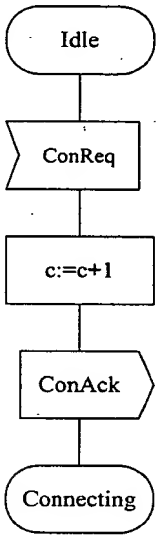
Anhang A

Figuren

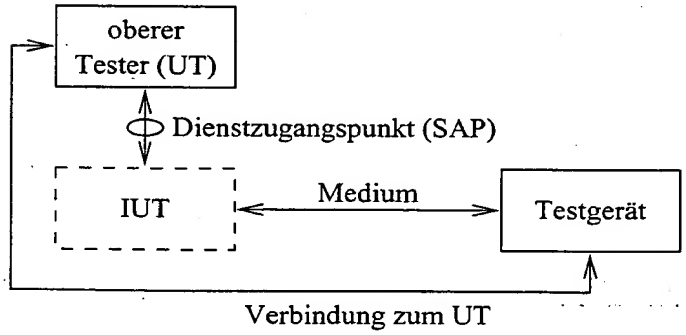
Abbildungsverzeichnis



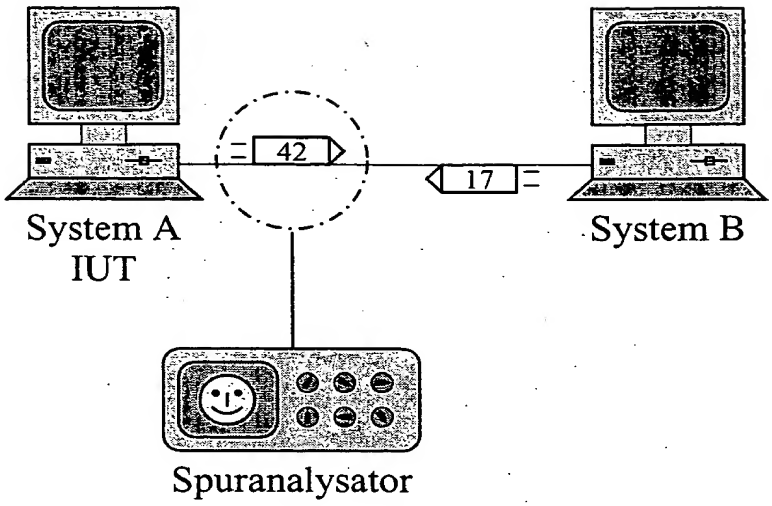
Figur A.1: Baumdarstellung des LTS für einen Getränkeautomaten.



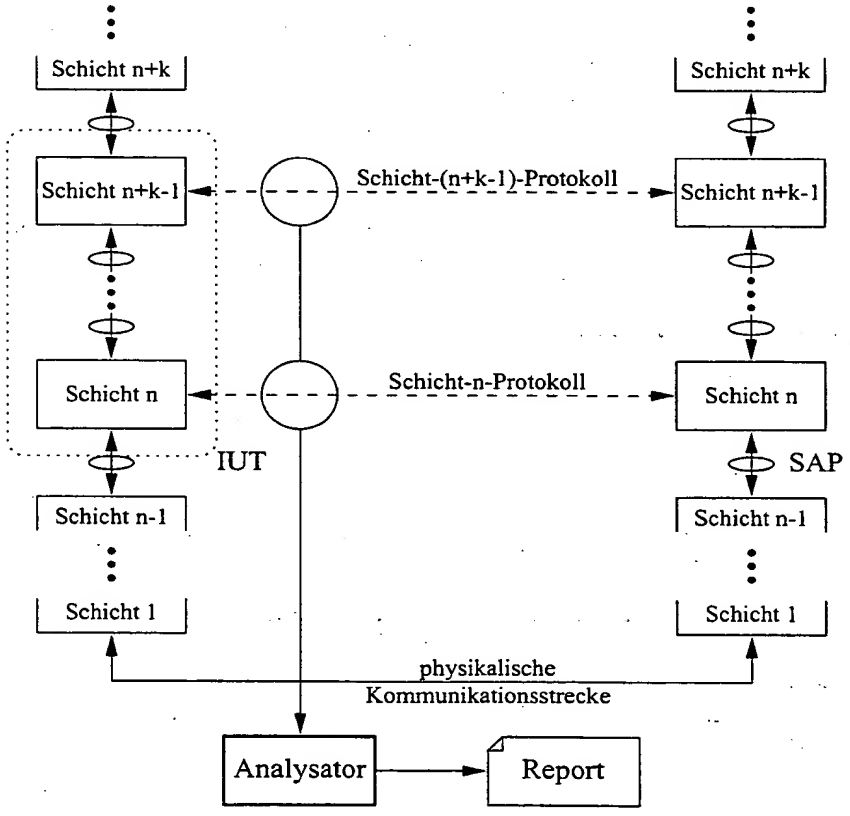
Figur A.2: Beispiel für eine Transition in SDL/GR.



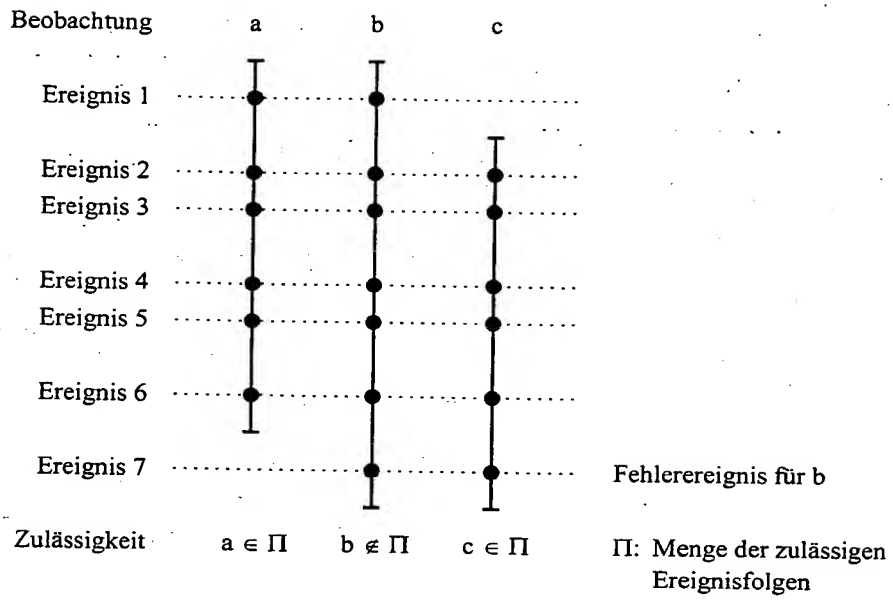
Figur A.3: Beispiel für eine Konfiguration zum Konformitätstest.



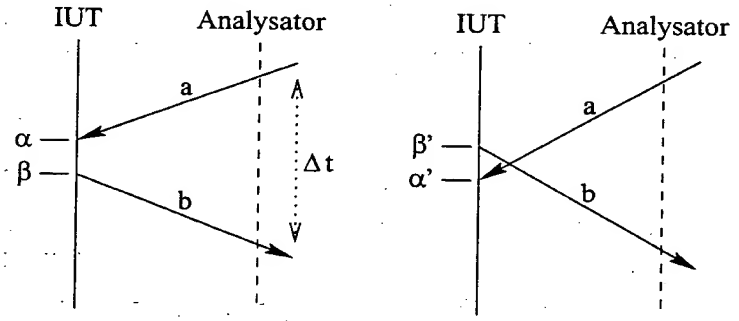
Figur A.4: Physische Einsatzkonfiguration des Spuranalysators.



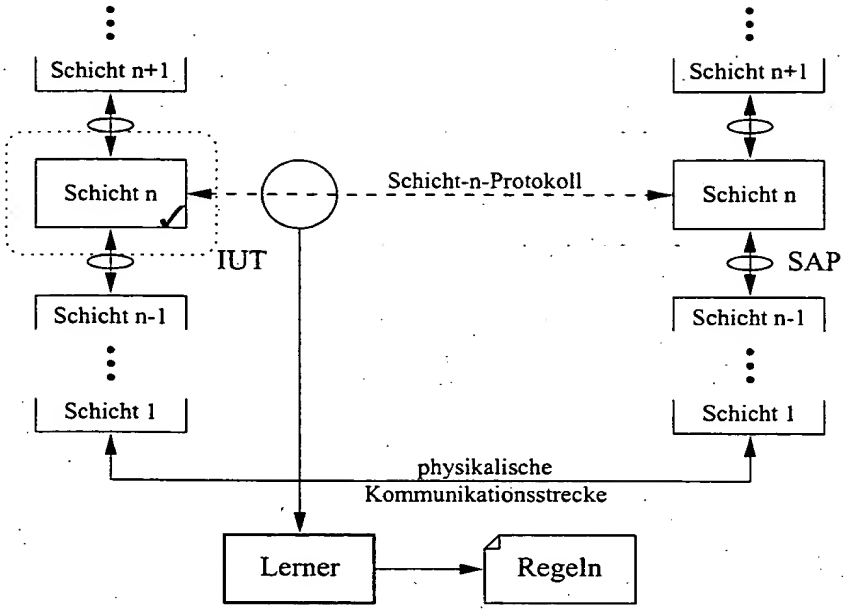
Figur A.5: Logischer Kontext der Spuranalyse.



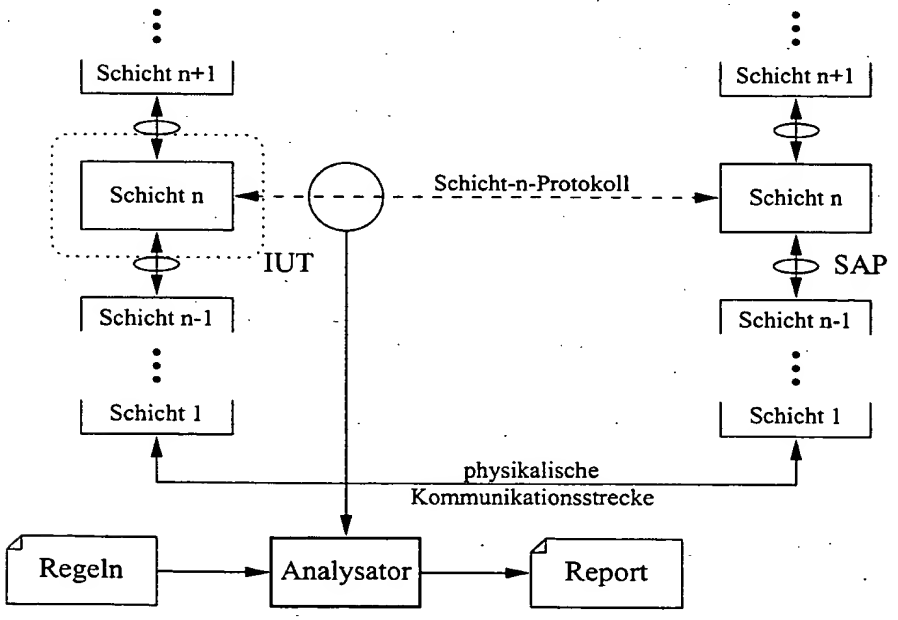
Figur A.6: Bedeutung von *Beobachtung* und *Fehlerereignis*. *c* ist ein möglicher Anfang für die neue Analyse nach dem Fehlerereignis.



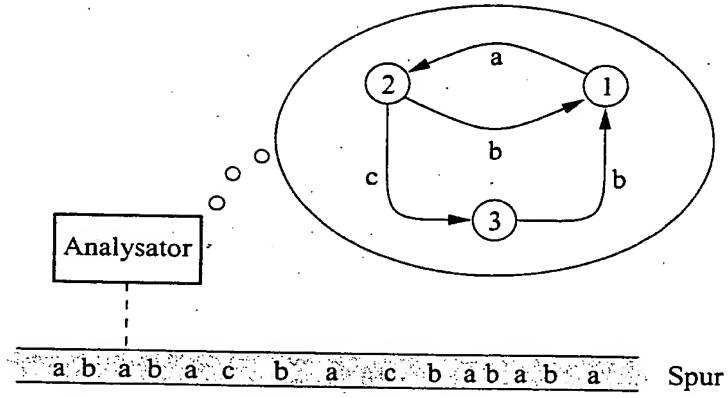
Figur A.7: Mehrdeutige Sequentialisierung der IUT-internen Aktionen α und β bei möglicher Nachrichtenkreuzung.



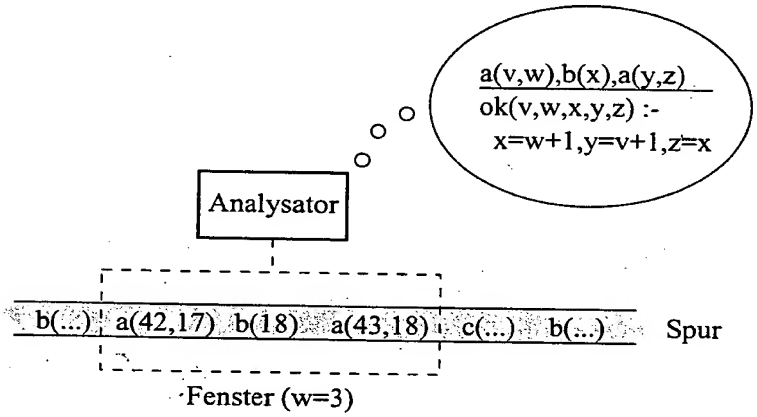
Figur A.8: Logisches Szenario der *Lernphase* eines lernfähigen Spuranalysators.



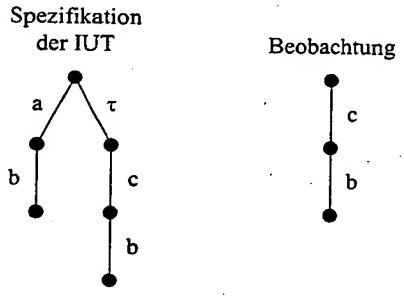
Figur A.9: Logisches Szenario der *Prüfphase* eines lernfähigen Spuranalysators.



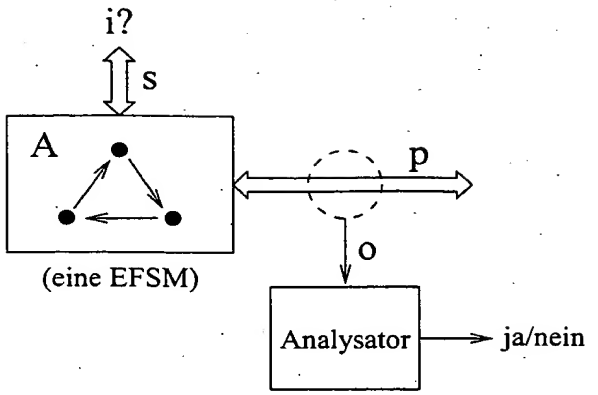
Figur A.10: Phase 1, Lernen der regulären Sprache der PDU-Typen.



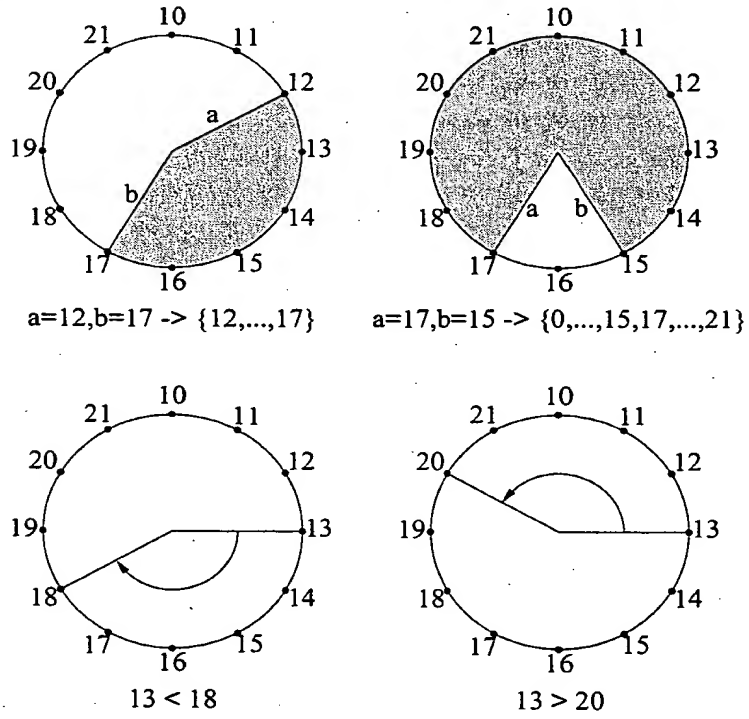
Figur A.11: Phase 2, Lernen des *ok*-Prädikats für die Nachrichtenattribute.



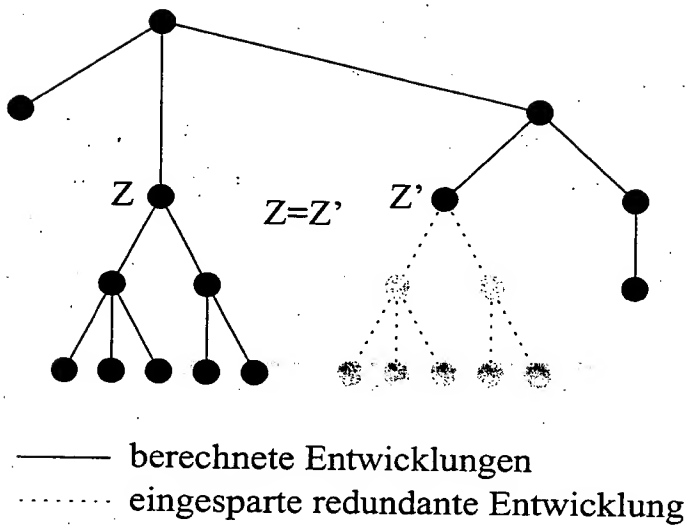
Figur A.12: Eine beispielhafte Protokollspezifikation als LTS und eine Beobachtung, deren Korrektheit von der Interpretation von a als Ein- oder Ausgabe abhängt.



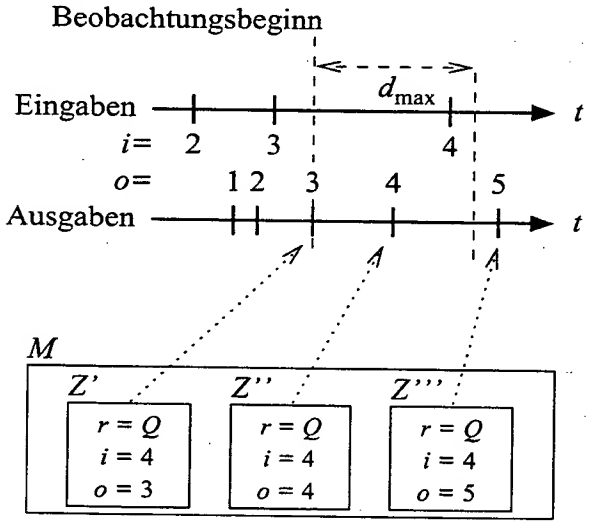
Figur A.13: Schema zur Definition der *vereinfachten Spuranalyse*.



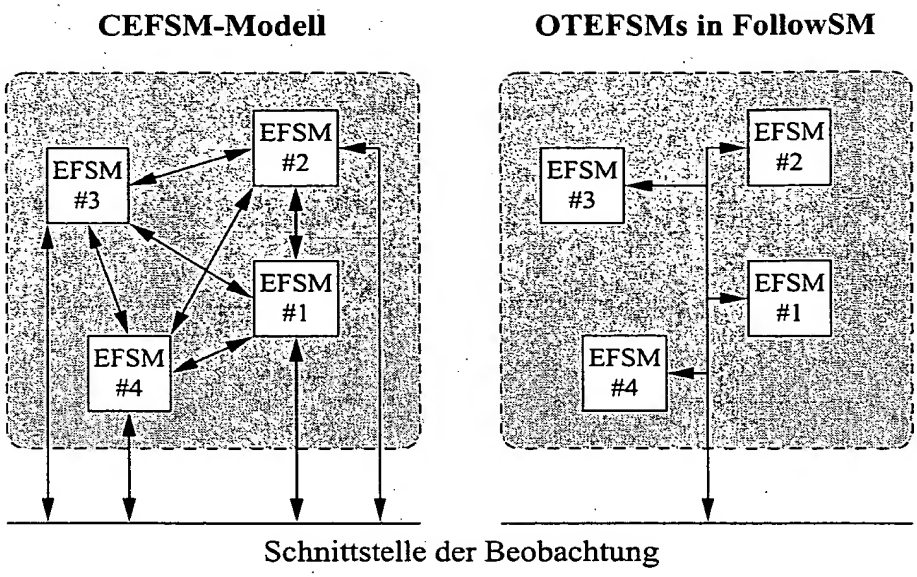
Figur A.14: Eigenschaften der Arithmetik auf Ringintervallen am Beispiel $Q = \{10, ..., 21\}$.



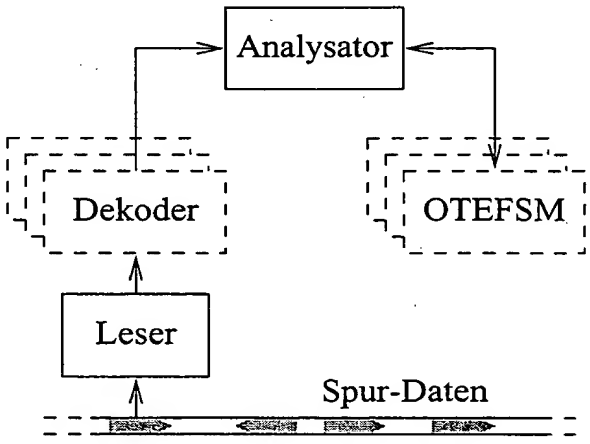
Figur A.15: Beispiel für eine Pfadvereinigung im Zustandssuchbaum bei Z und Z'.



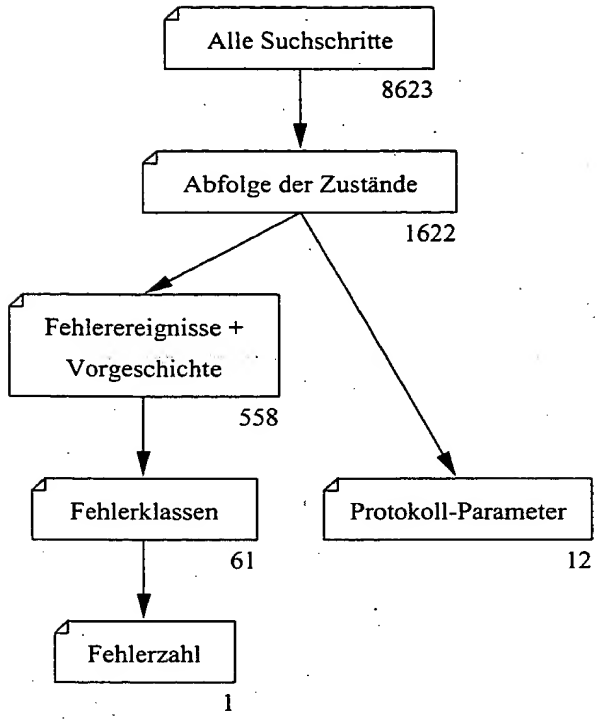
Figur A.16: Bildung der initialen Zustandsliste mit allen in Frage kommenden Ausgabe-Indizes.



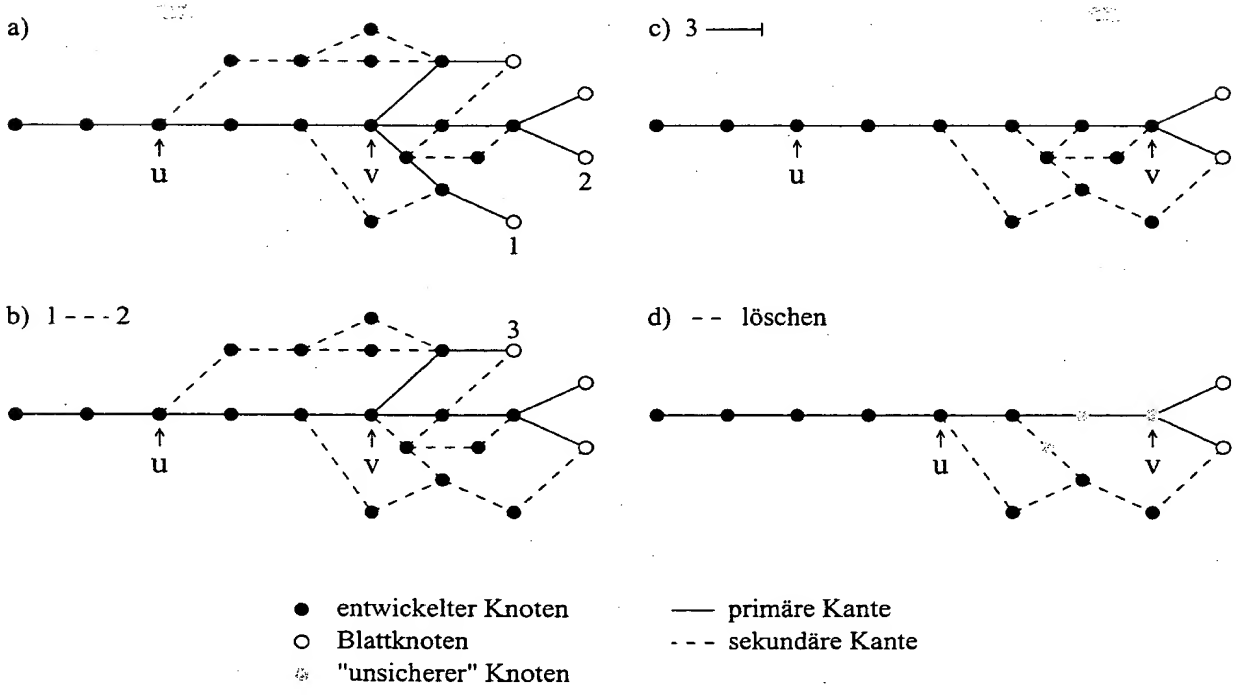
Figur A.17: Mehrere Automaten im CEFSM-Modell (links) und im OTEFSM-Modell von *FollowSM* (rechts).



Figur A.18: Datenfluß zwischen den wichtigsten Teilsystemen in der *FollowSM*-Implementierung (gestrichelt = protokollspezifisch, durchgezogen = protokollunabhängig).



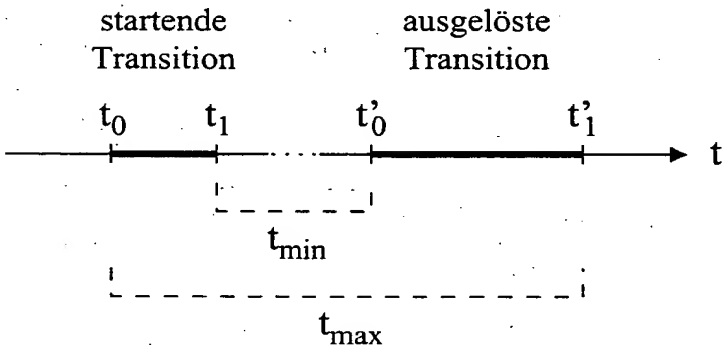
Figur A.19: Hierarchie der *FollowSM*-Analyseberichte.



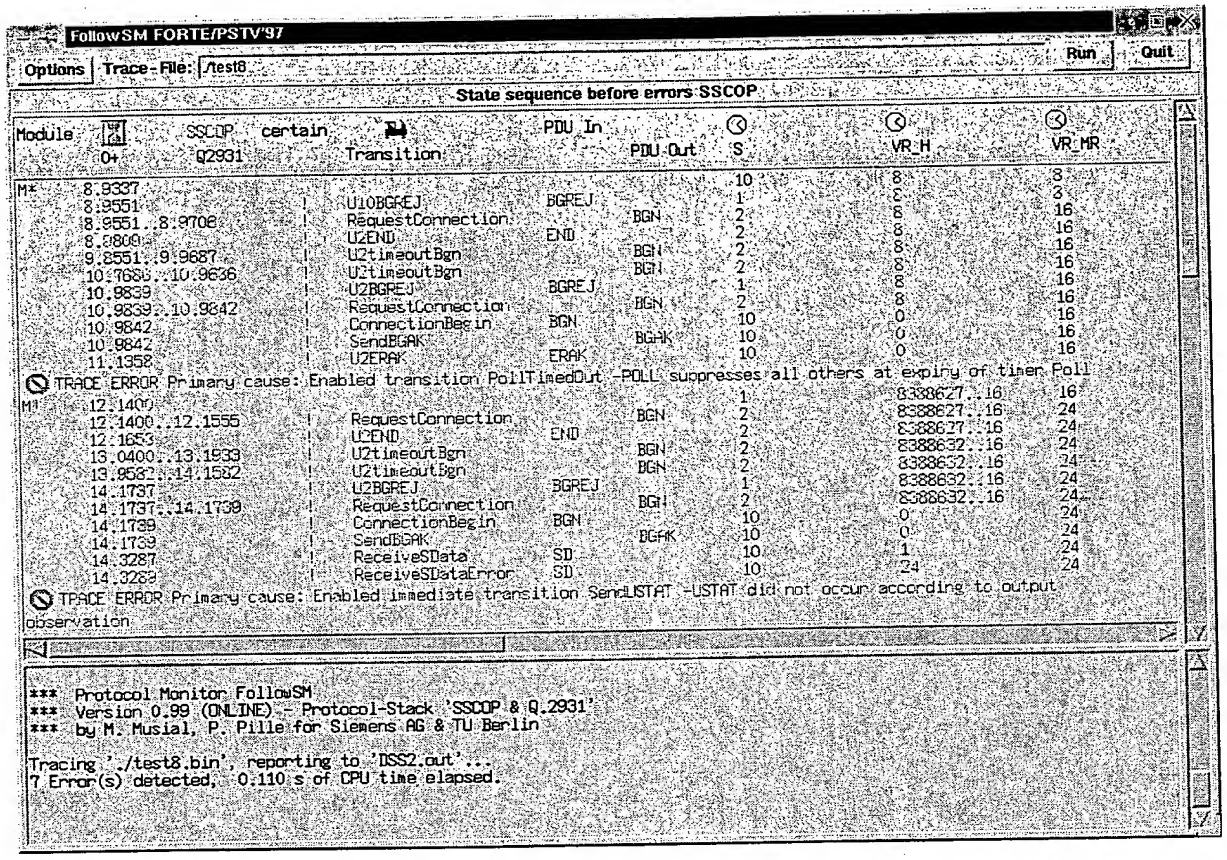
Figur A.20: Beispiel für Operationen auf der *Pfad*-Datenstruktur zur Rekonstruktion einer erklärenden Zustandssequenz.

Time	Transition	PDU in	PDU out	State	Timer_CC	VT(CC)
0.0000				?	???	???
0.0000	SendENDAK		ENDAK	?	???	???
0.0250	U3BGAK	BGAK		3..10	???	???
0.0250..0.0252	U4TimedOut			1	off	4..-96
0.0250..0.0252	RequestConnection		BGN	2	0.9250..4.0252	1
0.0252	U2END	END		2	0.9250..4.0252	1
0.0747	U2BGREJ	BGREJ		1	off	1
0.0747..0.0749	RequestConnection		BGN	2	0.9747..4.0749	1
0.0749	ConnectionBegin	BGN		10	off	1

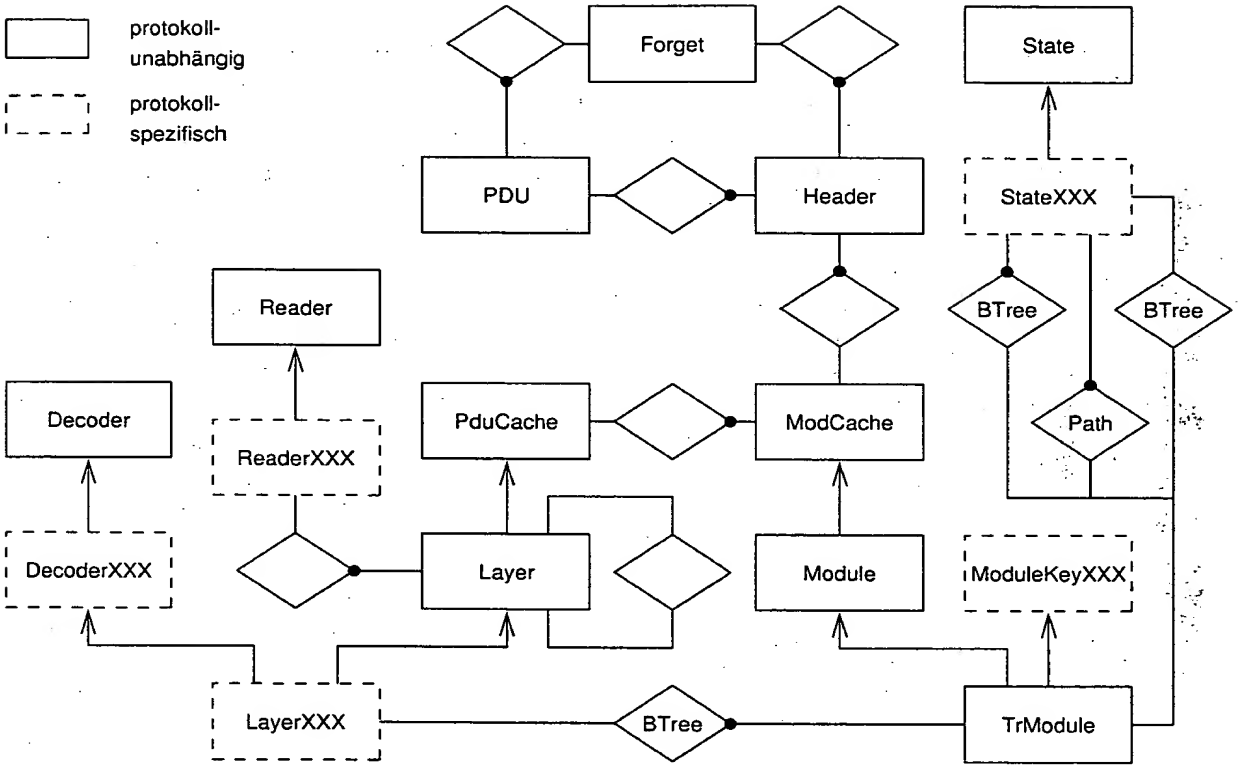
Figur A.21: Beispiel für die Struktur des Analyseberichts *Zustandsfolge* in einer Synchronisationsphase.



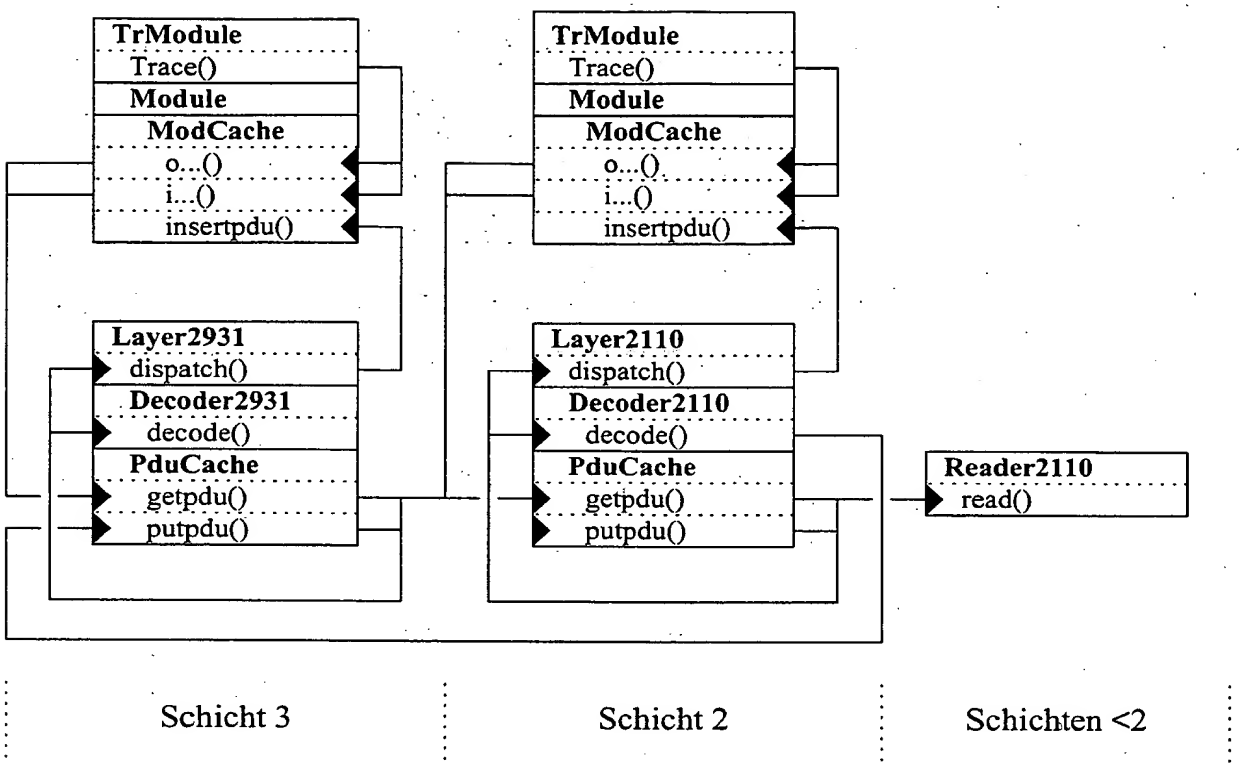
Figur A.22: Rekonstruktion der minimalen und maximalen Zeitgeberlaufzeit.



Figur A.23: Bildschirmanzeige der Benutzeroberfläche für den *FollowSM*-Prototyp.



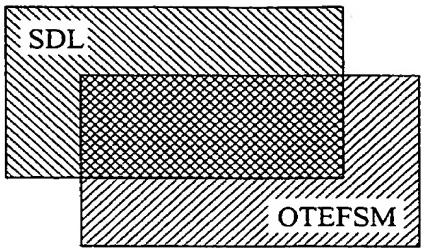
Figur A.24: Klassenhierarchie und Informationsmodell.



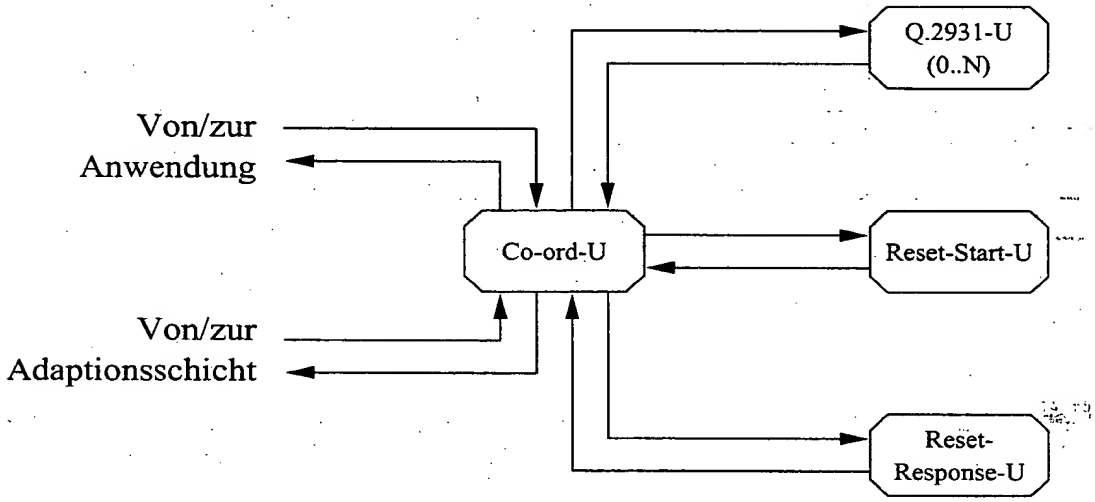
Figur A.25: Aufrufdiagramm der PDU-Verwaltung am Beispiel der UNI-Signalisierung im B-ISDN.

```
// SendRs - ITU-T Q.2110-U 20/51
Defn_Pre(sendrs) {
    SingleR(adproc,eq,send_RS);
}
Defn_NPre(sendrs) {
    SingleR(adproc,ne,send_RS);
}
Defn_Post(sendrs) {
    OUTPUT(RS);
    AndR(vr_h,le,op.n_mr);
    LastR(vt_sq,eq,op.n_sq);
}
Defn_Trans(sendrs) {
  OUTPUT(RS);
  vr_mr = op.n_mr;
  adproc=0;
}
```

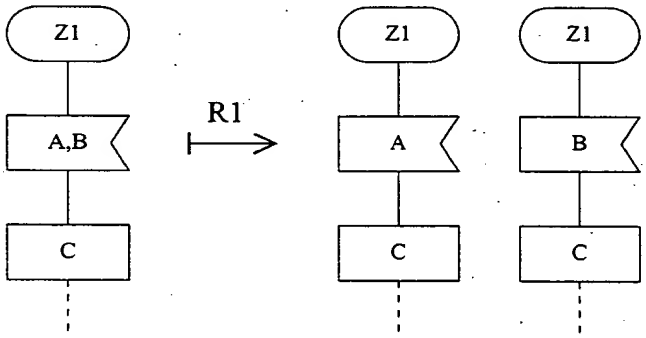
Figur A.26: Beispiel für den C++-Programmtext einer Transition.



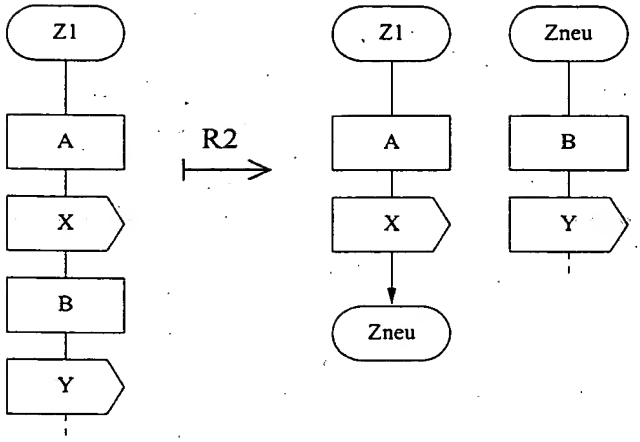
Figur A.27: Relation der Modellierungsmächtigkeiten von SDL und dem OTEFSM-Modell.



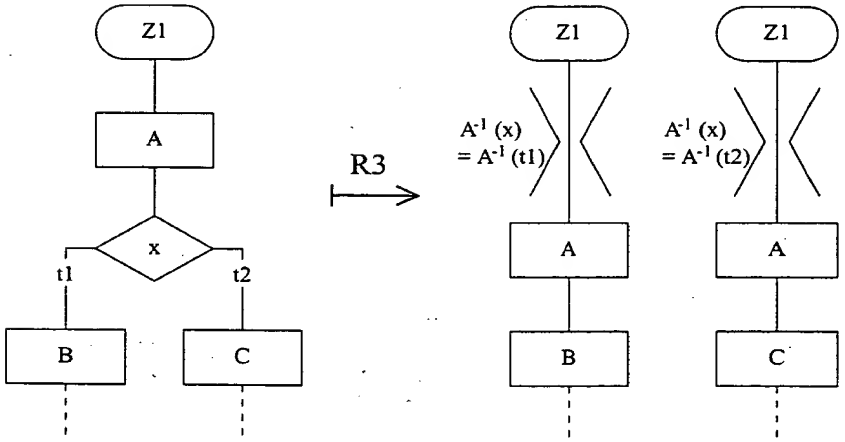
Figur A.28: SDL-Prozeßstruktur aus der Q.2931-Spezifikation [IT94b]. Grau: Prozesse, die OTEFSM-Module des abgeleiteten Analysators ergeben.



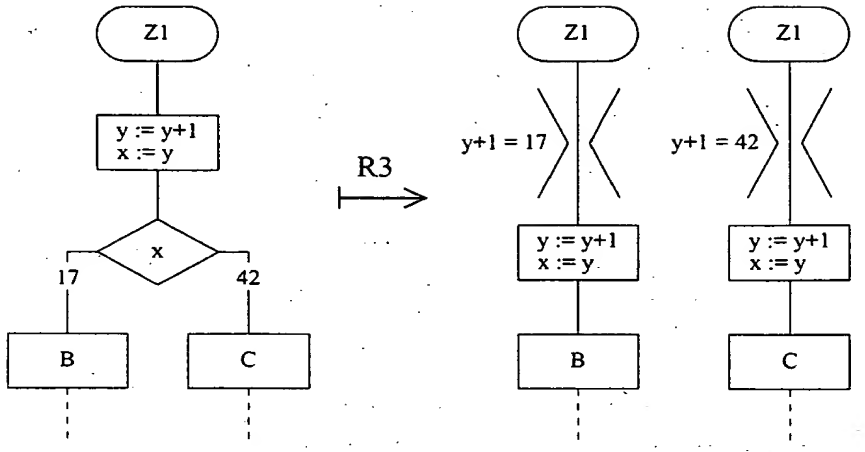
Figur A.29: Transformation von SDL-Transitionen mit mehreren Stimuli (A, B).



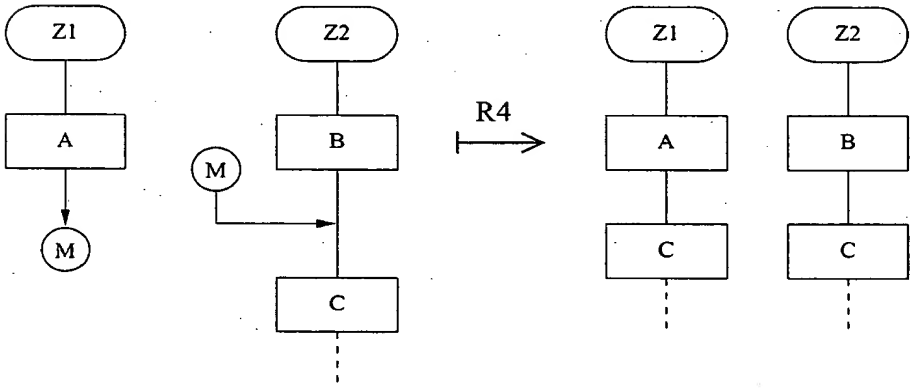
Figur A.30: Transformation von SDL-Transitionen mit mehreren Ausgaben (X, Y).



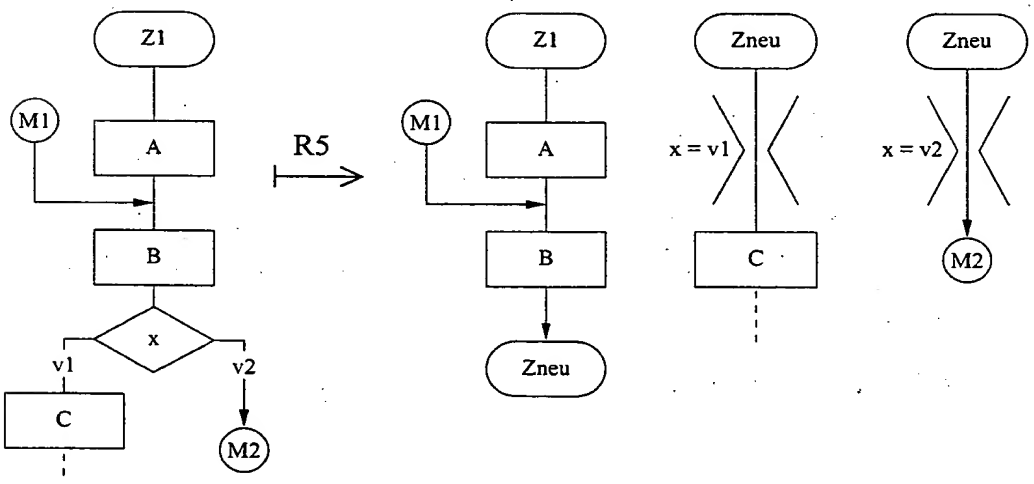
Figur A.31: Transformation von SDL-Transitionen mit Verzweigung.



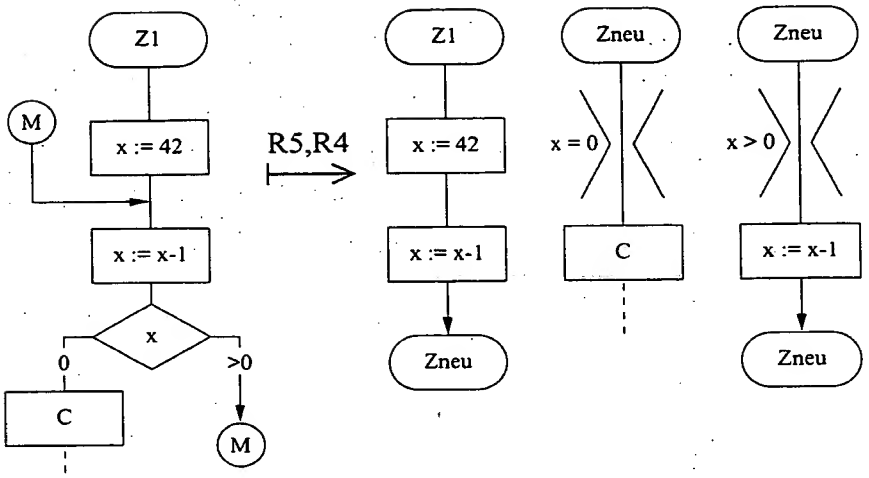
Figur A.32: Beispiel für die Anpassung der Vorbedingung.



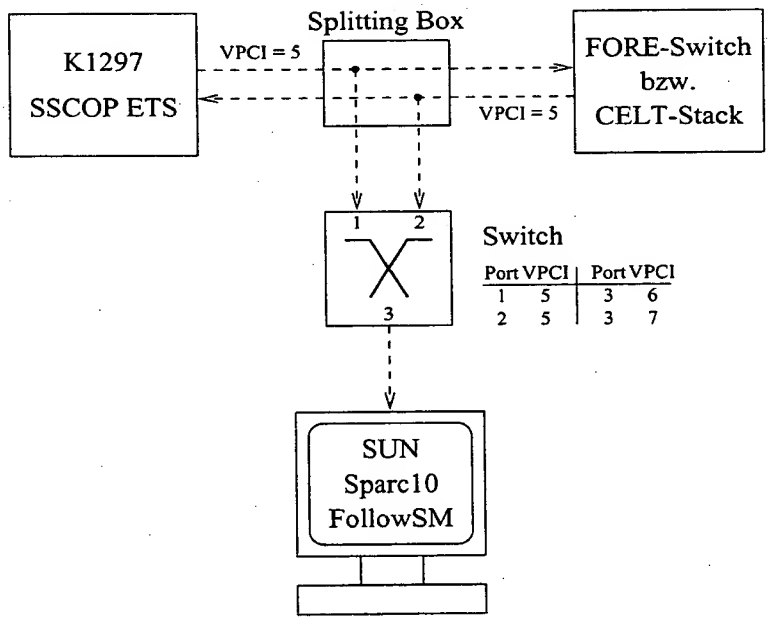
Figur A.33: Transformation von SDL-Transitionen mit Sprüngen.



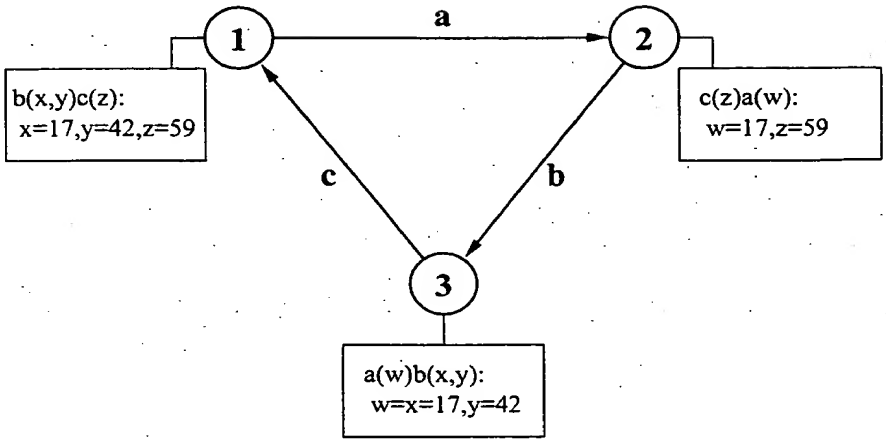
Figur A.34: Transformation von SDL-Transitionen mit Verzweigung nach einer Marke.



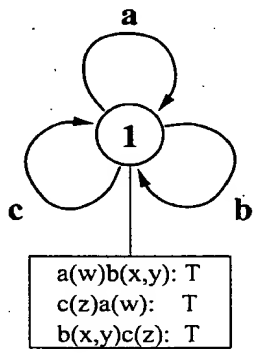
Figur A.35: Beispiel für die Übersetzung einer Schleife mittels R4 und R5.



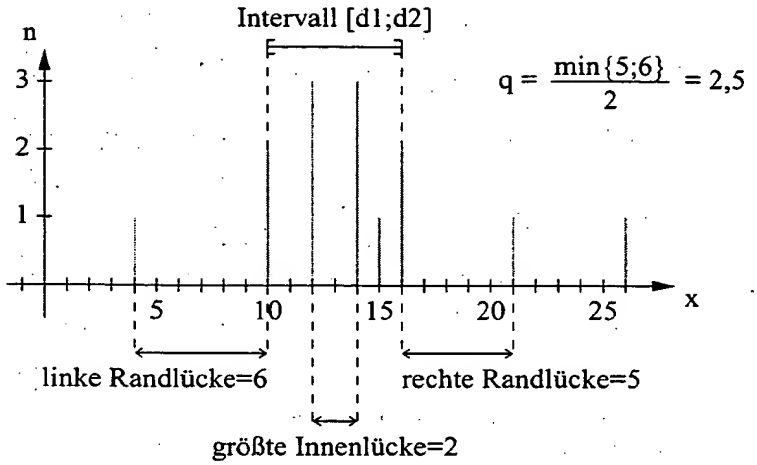
Figur A.36: Versuchsaufbau des „Online"-Tests.



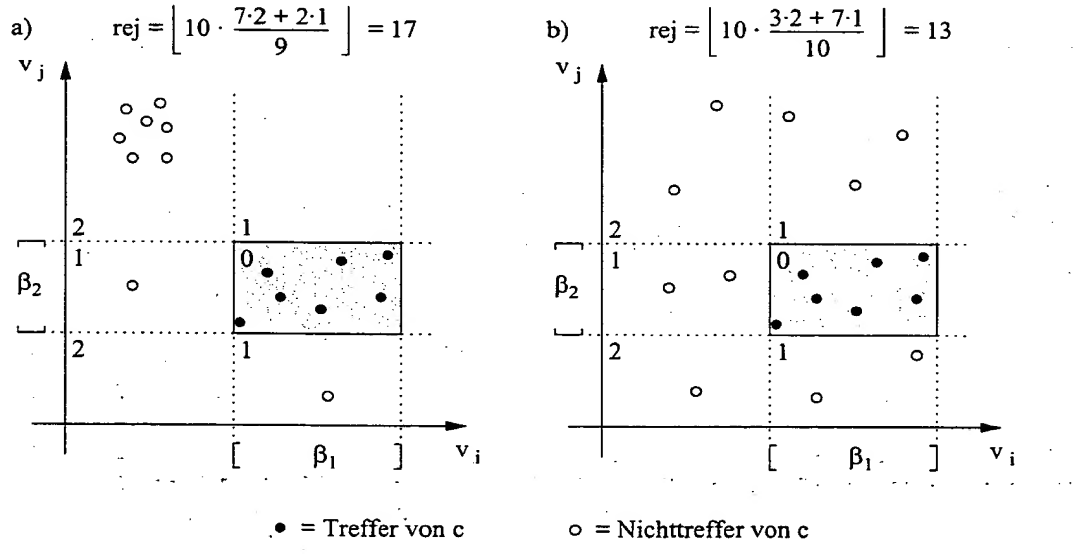
Figur A.37: Der Automat A_{min}, der nur das Lehrbeispiel $a(17), b(17, 42), c(59)$ akzeptiert.



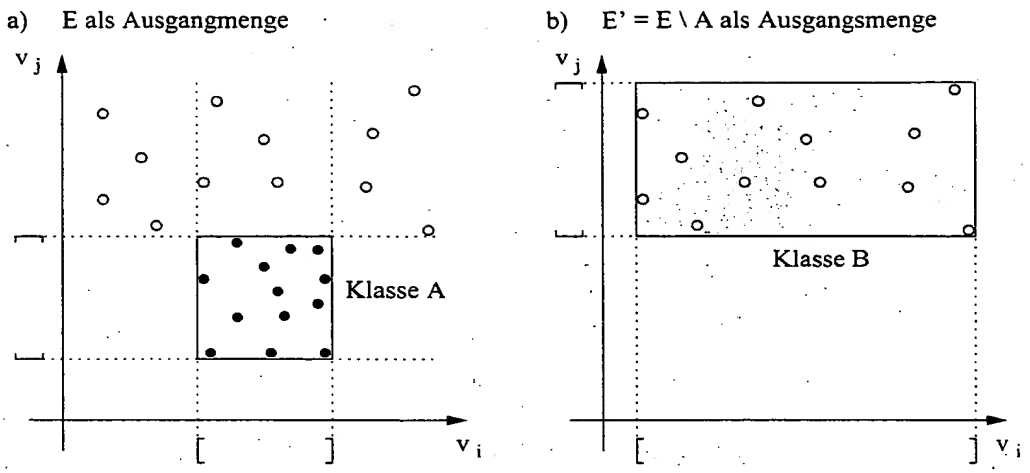
Figur A.38: Der Automat A_{max}, der alle möglichen PDU-Folgen zum Lehrbeispiel $a(17), b(17, 42), c(59)$ akzeptiert.



Figur A.39: Beispiel für die Berechnung des Häufungsquotienten q. Die dargestellte Wahl von $[d_1; d_2]$ maximiert q, das Intervall [14; 16] würde nur $q = 2$ liefern!



Figur A.40: Berechnung der Komponente *rej* der Klausel-Qualitätsfunktion. Die Zahlen geben die Anzahl der „Nichttreffer" pro Trainingsbeispiel an.



Figur A.41: Beispiel für die Auswirkung der Beispiel-Vorauswahl: B läßt sich erst nach Ausblenden von A diskriminieren.

```
-- 10 Beispiele Klasse 1
v1 = v3 * v4 = 0..282
v2 = v3 + v4 = 37..53
v3 = 37..47
v4 = 0..6 = v2 - v3 = v2 - 37..47

-- 20 Beispiele Klasse 2
v1 = 90..110
v2 = -30..30
v3 = v1 + v2 + -3..3 = 60..140
v4 = v3 - v1 + -3..3 = v2 + -6..6
```

```
ok (v1,v2,v3,v4) <==
       v4 * v3 - v1 = 0
  AND  v4 - v2 + v3 = 0
  AND  -4 <= v3 - v2 <= 0
  AND  0 <= v4 <= 4
  AND  40 <= v2 <= 48
[  10/1/294089]

ok (v1,v2,v3,v4) <==
       -3 <= v3 - v2 - v1 <= 3
  AND  -6 <= v4 - v2 <= 5
  AND  -3 <= v3 - v1 - v4 <= 3
  AND  92 <= v1 <= 110
  AND  -30 <= v2 <= 27
[  20/2/198089]
```

Figur A.42: *LARGE* auf Beispieldatensatz 1. Links die Regeln für die Datenerzeugung, rechts das Lernergebnis.

```
ok (v1,v2,v3,v4) <==
       v4 * v3 - v1 = 0
  AND  v3 + v4 - v2 = 0
  AND  37 <= v3 <= 47
  AND  -47 <= v4 - v2 <= -37
[ 100/1/252086]

ok (v1,v2,v3,v4) <==
       -113 <= v2 - v3 <= -87
  AND  -6 <= v4 - v2 <= 6
  AND  -113 <= v4 - v3 <= -88
  AND  -3 <= v3 - v2 - v1 <= 3
  AND  -3 <= v4 + v1 - v3 <= 3
[ 200/1/174885]
```

Figur A.43: Lernresultat von Datensatz 1 mit 300 Beispielen.

```
-- 25 Beispiele Klasse 1
v1 = 0..999
v2 = 0..999
v3 = v1 - v2

-- 25 Beispiele Klasse 2
v1 = 0..999
v2 = v1 + v3
v3 = 0..999

-- 25 Beispiele Klasse 3
v1 = v2 * v3
v2 = 0..999
v3 = 0..999

-- 25 Beispiele Klasse 4
v1 = 42
v2 = 0..999
v3 = -v2
```

v_1	v_2	v_3
840	1193	353
318	1204	886
584	159	425
691	749	58
42	163	-163
533	604	-71
104910	269	390
742	1040	298
404	857	-453
42	662	-662
42	2	-2
532	1319	787
⋮	⋮	⋮

```
ok (v1,v2,v3) <==
        v1 = 42
  AND   v3 + v2 = 0
  AND   v3 - v1 + v2 = -42
  AND   -35448 <= v3 * v1 <= -84
[  25/1/296489]

ok (v1,v2,v3) <==
        v2 * v3 - v1 = 0
  AND   6528 <= v1 <= 729708
  AND   -728832 <= v3 - v1 <= -6504
  AND   -728875 <= v2 - v1 <= -6256
[  25/1/128089]

ok (v1,v2,v3) <==
        v2 - v3 - v1 = 0
  AND   434 <= v3 + v1 <= 1900
  AND   65 <= v1 <= 982
  AND   2840 <= v1 + 6 * v2 <= 12365
[  25/1/108886]

ok (v1,v2,v3) <==
        v3 + v2 - v1 = 0
  AND   44 <= v3 + v2 <= 947
  AND   44 <= v1 <= 947
  AND   -393 <= v3 + v1 <= 1702
[  25/1/ 90089]
```

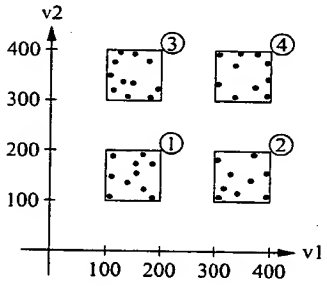
Figur A.44: *LARGE* auf Beispieldatensatz 2. Links die Regeln für die Datenerzeugung, rechts das Lernergebnis.

```
-- 100 Beispiele Klasse 1
v1 = v3 * v4 = 0..282
v2 = v3 + v4 = 37..53
v3 = 37..47
v4 = 0..6 = v2 - v3 = v2 - 37..47
```

```
ok (v1,v2,v3,v4) <==
       v3 * v4 - v1 = 0
  AND  v3 + v4 - v2 = 0
  AND  0 <= v4 <= 6
  AND  -6 <= v3 - v2 <= 0
  AND  38 <= v2 <= 53
[ 100/0/203089]
```

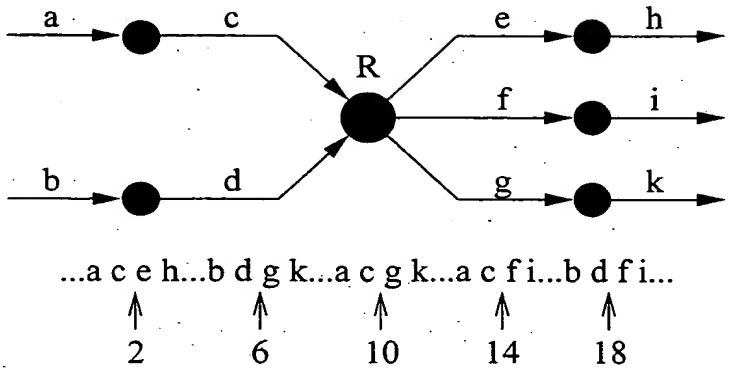
Figur A.45: *LARGE* auf Beispieldatensatz 3. Regeln für die Datenerzeugung und Lernergebnis.

```
-- 100 Beispiele Klasse 1
v1 = 100..200
v2 = 100..200

-- 100 Beispiele Klasse 2
v1 = 300..400
v2 = 100..200

-- 100 Beispiele Klasse 3
v1 = 100..200
v2 = 300..400

-- 100 Beispiele Klasse 4
v1 = 300..400
v2 = 300..400
```



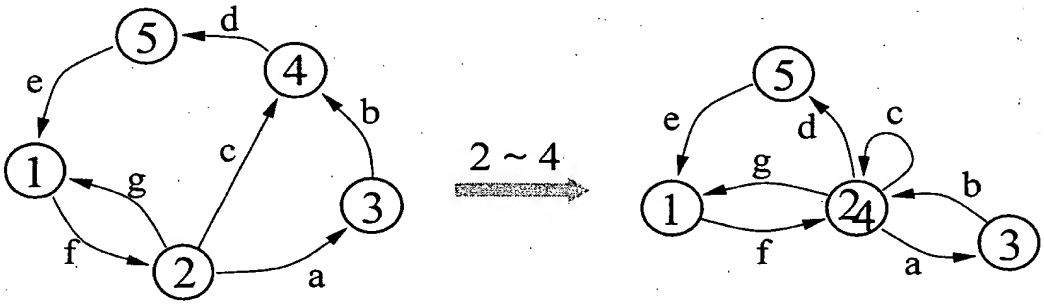

```
ok (v1,v2) <==
       300 <= v2 <= 400
  AND  101 <= v1 <= 200
  AND  115 <= v2 - v1 <= 286
[ 100/2/101295]

ok (v1,v2) <==
       300 <= v1 <= 399
  AND  300 <= v2 <= 399
  AND  -92 <= v2 - v1 <= 82
[ 100/2/ 80095]

ok (v1,v2) <==
       100 <= v2 <= 199
  AND  100 <= v1 <= 199
  AND  -94 <= v2 - v1 <= 77
[ 100/2/ 72095]

ok (v1,v2) <==
       301 <= v1 <= 399
  AND  101 <= v2 <= 199
  AND  -283 <= v2 - v1 <= -109
[ 100/2/ 41495]
```

Figur A.46: *LARGE* auf Beispieldatensatz 4. Regeln für die Datenerzeugung, Attributraum und Lernergebnis.

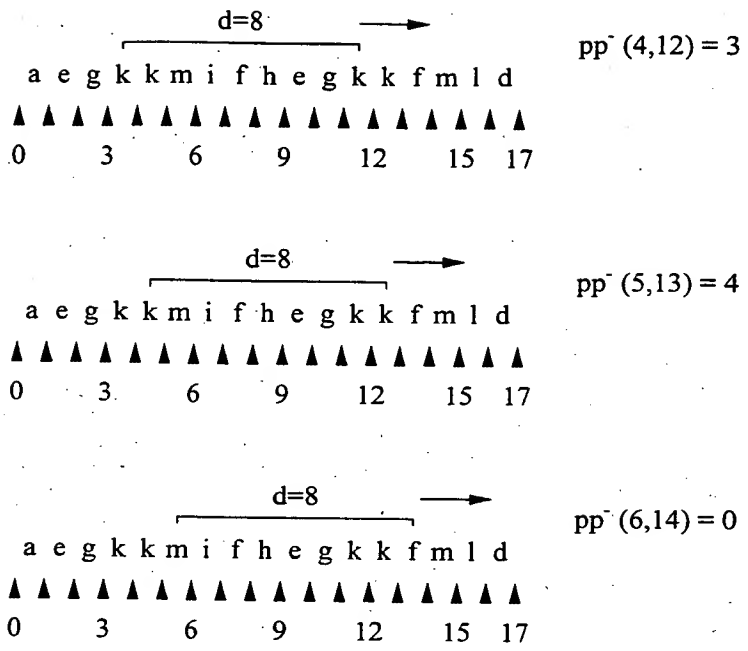


2~10, 10~6, 6~18, 18~14,14~2 => R={2;6;10;14;18}

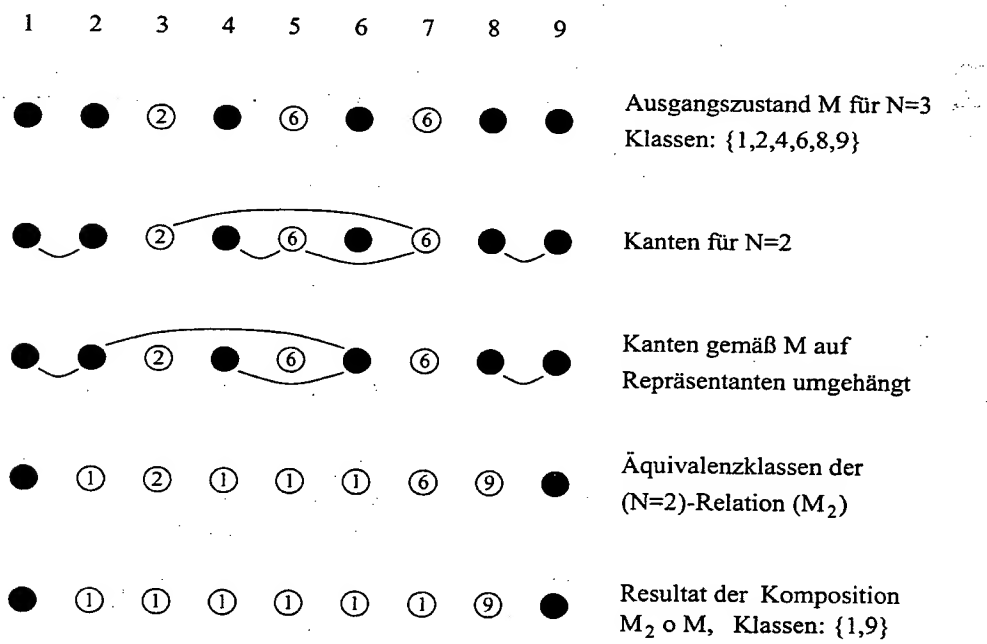
Figur A.47: Beispiel für die Erzeugung der zustandsbildenden Äquivalenzklassen von Sequenzpositionen in *InfSM*.



Figur A.48: Beispiel für den Einfluß der Ähnlichkeitsschwelle auf den von *InfSM* gebildeten Automaten.



Figur A.49: Berechnung der gemeinsamen Präfixlängen in *InfSM* mit quadratischen Aufwand.



Figur A.50: Berechnung der nächstgröberen Zerlegung in *InfSM* mit allen Teilschritten. Für schwarze Knoten gilt $M(i) = i$ (Repräsentantenknoten einer Klasse), sonst ist $M(i)$ im Knoten vermerkt.

Algorithmus *Lerne*(p, w)
$p = p_1 p_2 \dots p_n$: $\Pi(\zeta)^*$
w : $\mathbb{N}$ Fenstergröße

InfSM$(a_1 a_2 \dots a_n) \rightarrow S_A, \delta_A, M$
Für $s \in S$
Für $\beta \in \Sigma^w$ mit $\beta = a_{i-w+1} \dots a_i$ in $p \ \wedge \ M(i) \sim s$
$E \leftarrow \{(x_1^1, \dots, x_1^{\kappa(a_1)}, \dots, x_w^1, \dots, x_w^{\kappa(a_w)})$
$\mid a_{i-w+1}(x_1^1, \dots, x_1^{\kappa(a_1)}), \dots, a_i(x_w^1, \dots, x_w^{\kappa(a_w)})$ in $p \ \wedge \ M(i) \sim s\}$
LARGE$(E) \rightarrow \mathrm{ok}(v_1, \dots, v_z)$
$\omega_A(s) \leftarrow \omega_A(s) \ \vee \ \mathrm{ok}(v_1, \dots, v_z)$

Ausgabe: A mit S_A, δ_A, ω_A

Ende von *Lerne*

Figur A.51: Skizze des aus *InfSM* und *LARGE* kombinierten Protokoll-Lernalgorithmus.

Algorithmus *InfSM**(α, N)

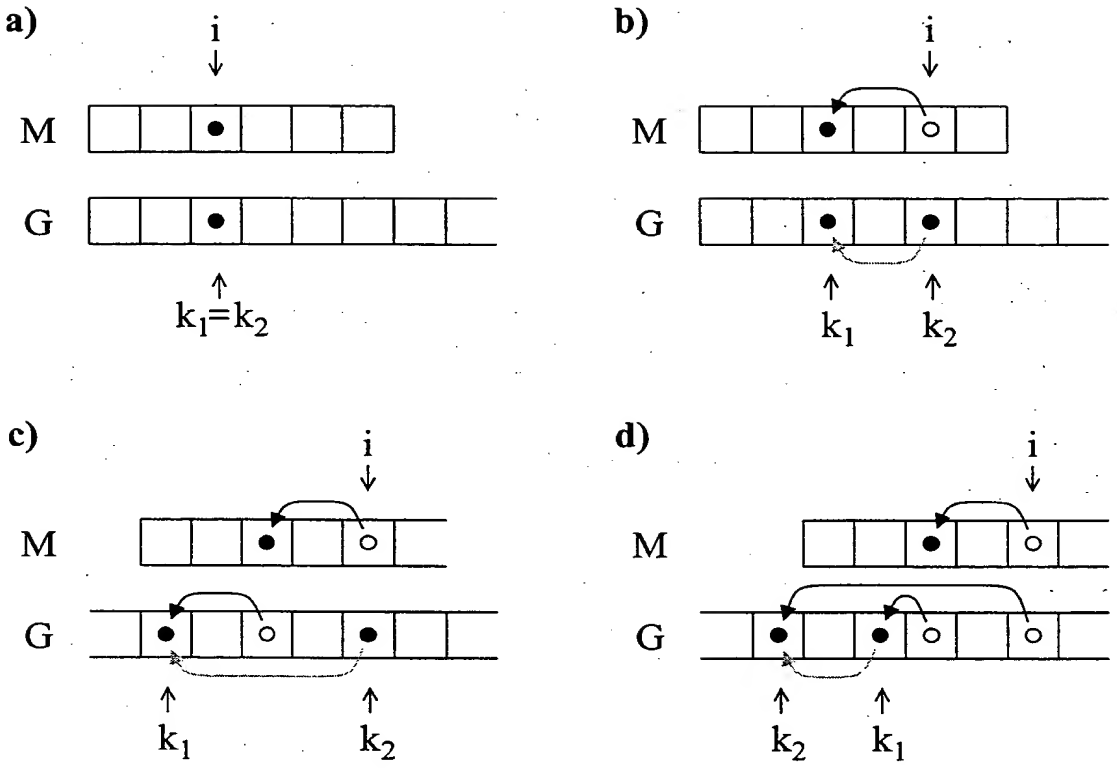
$\alpha = a_1 \dots a_n$: Σ^* -- Eingabesequenz
N : $\mathbb{N}$ -- Fenstergröße

G : $0 \dots n \to 0 \dots n$ -- globale Zuordnungsfunktion

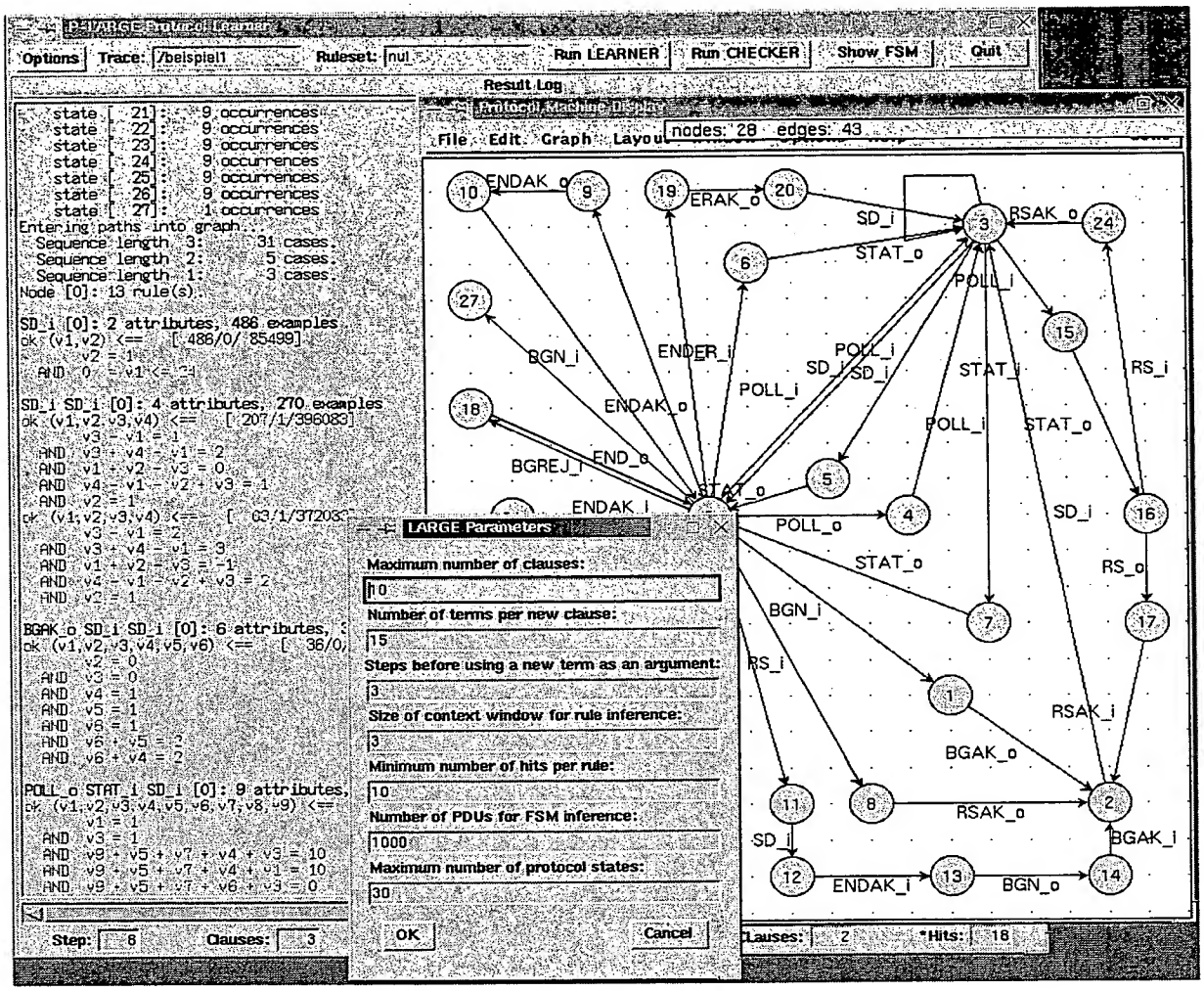
$G \leftarrow \mathrm{id}_{0\dots n}$
$p \leftarrow 1$
Wiederhole
  $\alpha' \leftarrow a_p \dots a_{p+N-1}$ -- Fenster in der PDU-Sequenz
  $M \leftarrow$ Zuordnungsfunktion $p \dots p+N-1 \to p \dots p+N-1$ gemäß *InfSM*(α')
  Für $i \in p \dots p+N-1$
    $k_1 \leftarrow M(i)$
    Solange $G(k_1) \neq k_1 : k_1 \leftarrow G(k_1)$ -- Zustand für $M(i)$ suchen
    $k_2 \leftarrow i$
    Solange $G(k_2) \neq k_2 : k_2 \leftarrow G(k_2)$ -- Zustand für i suchen
    $G(\max\{k_1; k_2\}) \leftarrow \min\{k_1; k_2\}$ -- zusammenfassen
  $p \leftarrow p + N/2$ -- Fenster verschieben
Bis α verarbeitet
Für $i \in 0 \dots n$
  $G(i) \leftarrow G(G(i))$ -- alle Verweise auf Repräsentanten

Ausgabe: Globale Zustands-Zuordnungsfunktion G
Ende von *InfSM**

Figur A.52: Algorithmus *InfSM** zur inkrementellen Anwendung von *InfSM* auf lange PDU-Sequenzen.



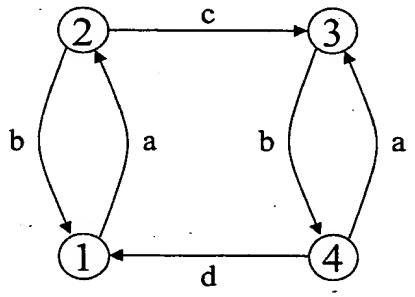
Figur A.53: Fälle, die bei der Zusammenfassung der Teilintervalle in *InfSM** auftreten. Graue Pfeile stehen für die jeweiligen Neueinträge.



Figur A.54: Beispiel für die Bildschirmanzeige des Prototyps des Protokoll-Lernprogramms.

```
LEDA.GRAPH
unknown
unknown
11
|{BGAK_i >
 { v1 22 = v2 0 = }
 { v1 11 = v2 0 = }
BGN_i >
 { v3 0 = v1 7 17 i v3 v1 + 7 17 i v2 1 78 i }
SD_i >
 { v2 1 = v1 0 24 i }
BGAK_i SD_i >
 { v1 22 = v4 v1 + 23 = v3 v1 + 22 = v2 v1 + 22 = v4 v3 + v1 + 23 = }
 { v1 11 = v4 v1 + 12 = v3 v1 + 11 = v2 v1 + 11 = v4 v3 + v1 + 12 = }
 .
 .
 .
1 8 0 |{RS_o}|
1 9 0 |{END_o}|
1 10 0 |{ER_i}|
1 11 0 |{ENDAK_o}|
2 1 0 |{STAT_i}|
3 1 0 |{STAT_o}|
4 1 0 |{RSAK_o}|
5 1 0 |{ENDAK_o}|
6 7 0 |{BGN_o}|
7 1 0 |{BGAK_i}|
7 7 0 |{ERAK_i}|
8 1 0 |{RSAK_i}|
9 1 0 |{BGREJ_i}|
10 1 0 |{ERAK_o}|
```

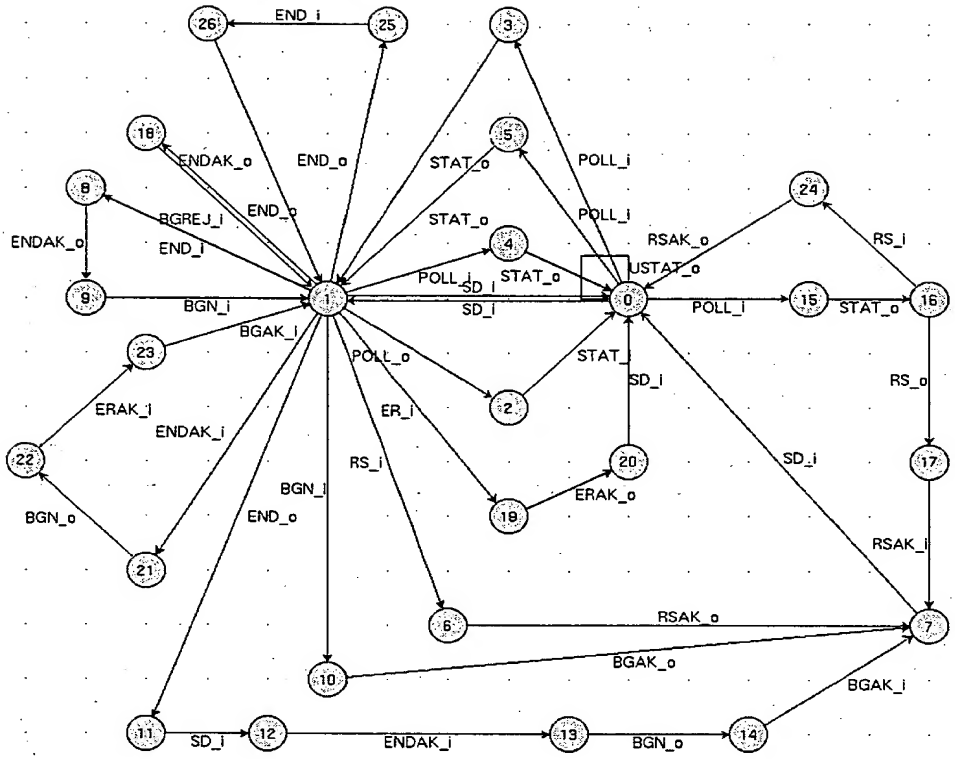
Figur A.55: Auszug aus der Repräsentation eines gelernten Protokollautomaten als LEDA-Graph.



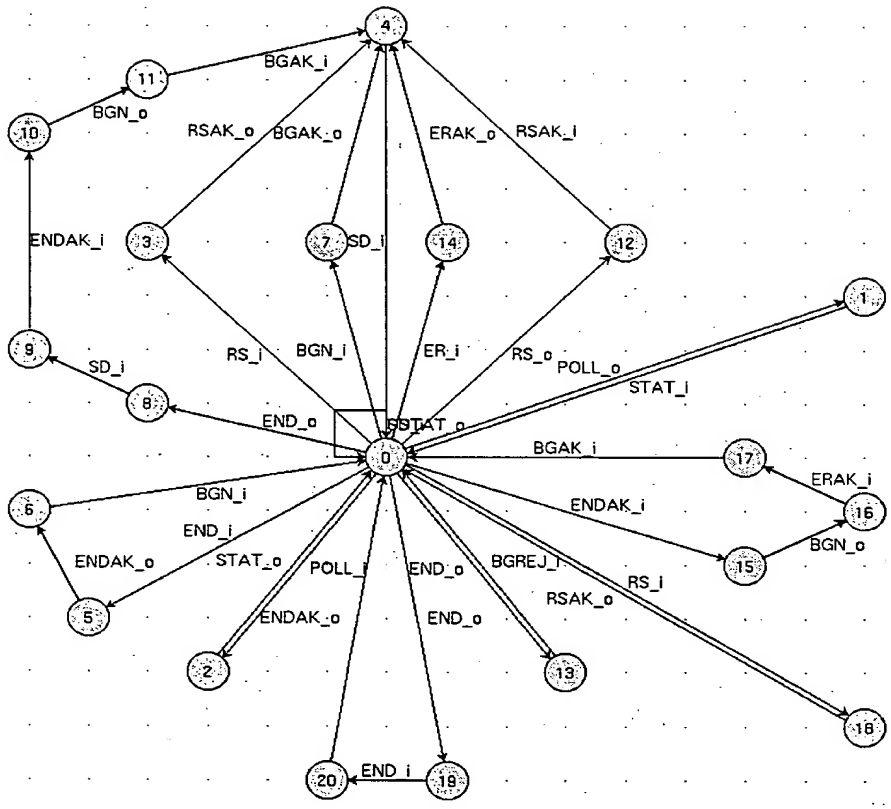
Beobachtung:

a b a b a b a b a b a b ...

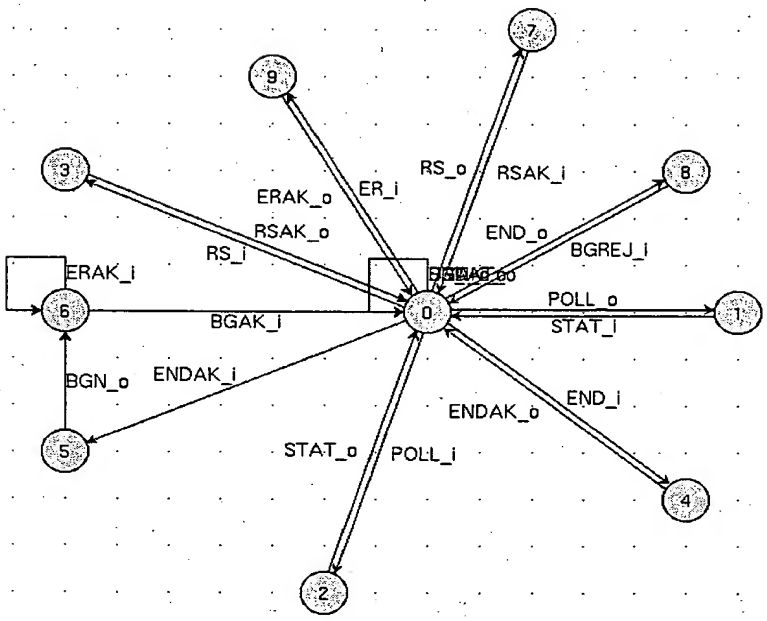
Figur A.56: Beispielsituation dafür, daß die Zustandsidentifikation selbst bei einem deterministischen endlichen Automaten unbeschränkt lange dauern kann.



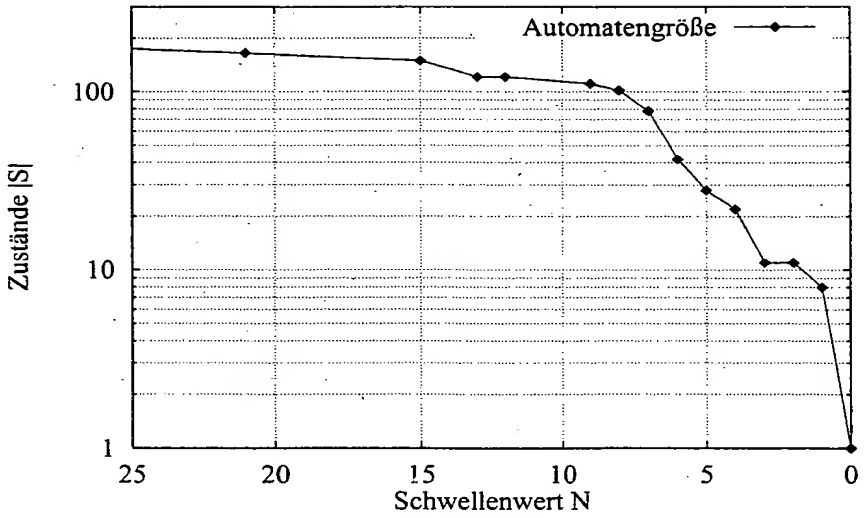
Figur A.57: Von *InfSM* abgeleiteter endlicher Automat der PDU-Typen mit 27 Zuständen für die Schwelle $N = 5$. Erzeugt aus einer Trainingsmenge von 1953 PDUs des SSCOP-Protokolls. Die Suffixe *_i* kennzeichnen Eingabe-, die *_o* Ausgabe-PDUs.



Figur A.58: Endlicher Automat aus derselben Trainingsmenge mit 21 Zuständen für die Schwelle $N = 4$.



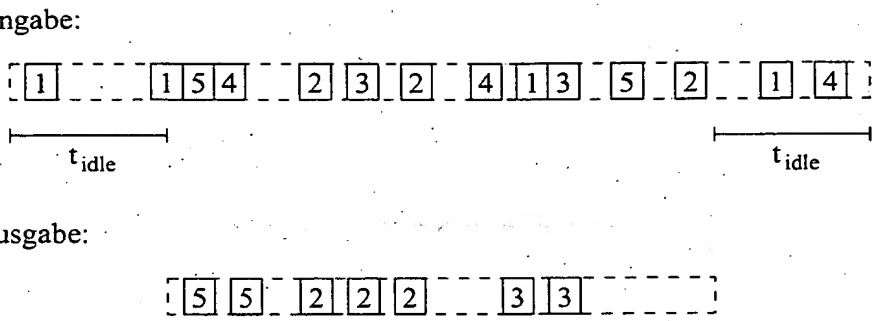
Figur A.59: Endlicher Automat aus derselben Trainingsmenge mit 10 Zuständen für die Schwellen $2 \leq N \leq 3$.



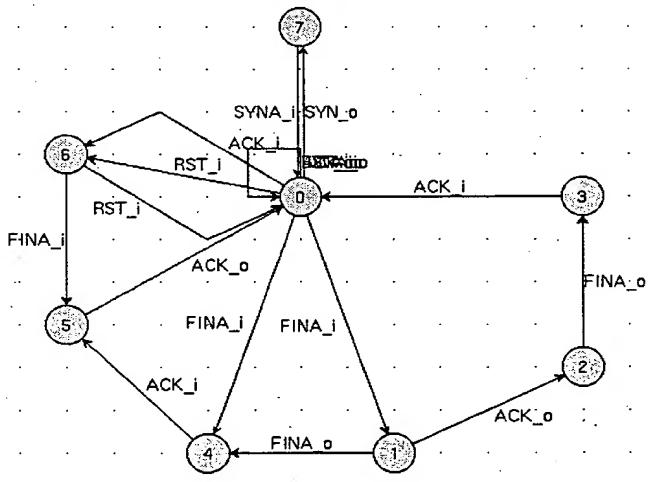
Figur A.60: Abnahme der Automatengröße $|S| = |\,\mathrm{range}(M_N)|$ in Abhängigkeit vom Schwellenwert N.

```
SD_i SD_i USTAT_o [0]: 8 attributes, 89 examples

ok (v1,v2,v3,v4,v5,v6,v7,v8) <==    [  63/1/666064]
        v8 - v4 - v6 = 0
  AND   v5 + v4 - v6 = 9
  AND   v5 + v6 - v8 - v3 = 6
  AND   v5 + v4 + v6 - v8 - v3 = 7
  AND   v8 - v4 - v6 - v1 + v5 = 9
  AND   v8 - v4 - v6 - v1 - v2 + v5 = 8
  AND   -16 <= v8 - v4 - v6 - v1 <= -1
  AND   -33 <= v8 - v4 - v6 - v1 - v7 <= -3
  AND   -17 <= v8 - v4 - v6 - v1 - v2 <= -2

ok (v1,v2,v3,v4,v5,v6,v7,v8) <==    [  26/1/576075]
        v8 - v4 - v6 = 1
  AND   v5 + v4 - v6 = 10
  AND   v5 + v6 - v8 - v3 = 5
  AND   v5 + v4 + v6 - v8 - v3 = 6
  AND   v8 - v4 - v6 - v1 + v5 = 11
  AND   v8 - v4 - v6 - v1 - v2 + v5 = 10
  AND   v2 = 1
  AND   v4 = 1
  AND   v8 - v5 = -7
```

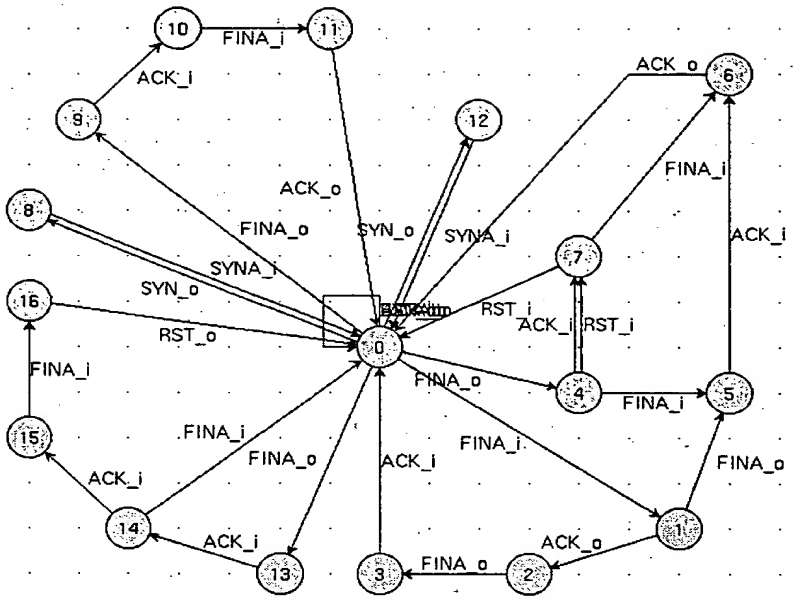
Figur A.61: Beispiel einer gelernten Attributregel für SSCOP.



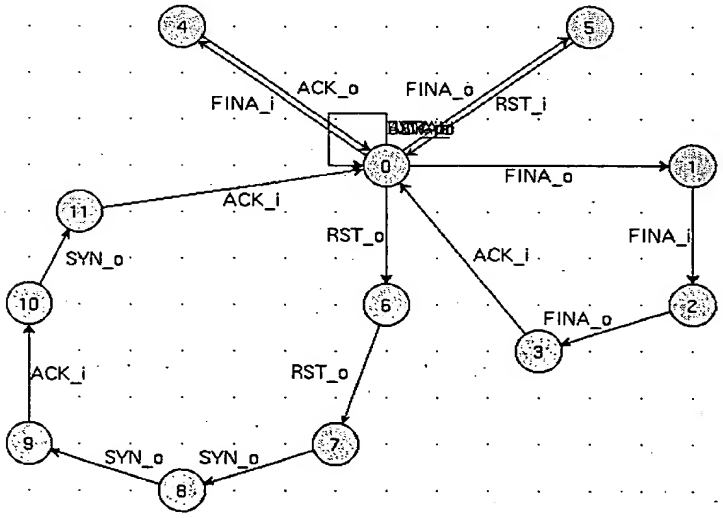
Figur A.62: Vorverarbeitung eines TCP-Mitschnitts mit mehreren gleichzeitigen Verbindungen zu einem geeigneten Lehrbeispiel mittels *tcp_split*. Die Ziffern kennzeichnen unterschiedliche Verbindungen.



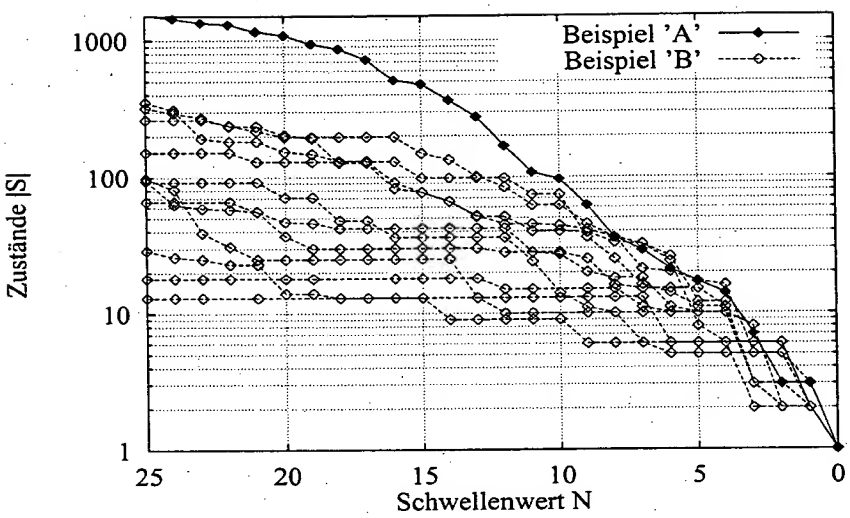
Figur A.63: Endlicher Automat für TCP nach Beispiel *B* mit 8 Zuständen.



Figur A.64: Endlicher Automat für TCP nach Beispiel *B* mit 17 Zuständen.



Figur A.65: Endlicher Automat für TCP nach Beispiel *A* mit 12 Zuständen.



Figur A.66: Abnahme der Automatengröße $|S|$ in Abhängigkeit vom Schwellenwert N für die TCP-Beispiele A und B. Jeder Graph entspricht einem *InfSM*-Lauf.

```
ACK_i FINA_i ACK_o [2]: 15 attributes, 40 examples

ok (v1,v2,v3,...,v13,v14,v15) <==   [  40/0/831694]
       v3 = 8760
  AND  v4 = 0
  AND  v8 = 8760
  AND  v9 = 0
  AND  v13 = 8760
  AND  v14 = 0
  AND  v11 - v1 = 0
  AND  v15 - v2 = 1
  AND  v12 - v5 = 0
  AND  v11 - v6 = 0
  AND  v15 - v7 = 1
  AND  v12 - v10 = 0
```

Figur A.67: Beispiel einer gelernten Attributregel für TCP.

Anhang B

Abkürzungen

Hier werden die wichtigen Abkürzungen aus dieser Arbeit aufgelistet, erklärt und auf die jeweilige Stelle der Einführung im Text verwiesen.

Abkürzung	Bedeutung	Seite
ATM	*Asynchronous transfer mode*, zellenbasierte Vermittlungstechnik für Breitbandnetze.	6
ASN.1	*Abstract syntax notation one*, Beschreibungssprache für abstrakte Syntax und Transfersyntax von Daten.	10
B-ISDN	Breitband-ISDN.	58
CCITT	*Comité Consultatif International Télégraphiqe et Téléphonique*, frühere Bezeichnung für den internationalen Dachverband der nationalen Netzbetreiber-Gesellschaften.	5
CCS	*Calculus of communicating systems*, Prozeßalgebra-Kalkül von Robin Milner.	7
CEFSM	*Communicating extended finite state machine*, kommunizierender erweiterter endlicher Automat.	58
CWA	*Closed world assumption*, Annahme, daß alle nicht in der Wissensbasis auftretenden Aussagen unerfüllt sind, bei Problemen des maschinellen Lernens.	17
DEA	*Deterministischer endlicher Automat.*	111
EFSM	*Extended finite state machine*, erweiterter endlicher Automat.	8
ETS	*Executable testsuite*, Satz von direkt ausführbaren Testsequenzen im Konformitätstest.	75
FollowSM	*Follow a state machine*, neuer Spuranalyse-Algorithmus mit vorgegebenem Protokollwissen.	29
FSM	*Finite state machine*, endlicher Automat.	8
GMD	*Gesellschaft für Mathematik und Datenverarbeitung*, deutsche Informatik-Forschungseinrichtung.	75
ILP	*Inductive logic programming*, induktive logische Programmierung, Paradigma zum Erlernen logischer Programme aus Beispielen.	16

Abkürzung	Bedeutung	Seite
InfSM	*Infer a finite state machine*, neuer Algorithmus zum Ableiten einer Familie endlicher Automaten aus einem Beispielwort.	104
IP	*Interaction point*, Schnittstelle einer getesteten Instanz zur Umgebung, d. h. zur nächsthöheren oder -niedrigeren Protokollschicht oder physikalische Verbindung zur Partnerinstanz.	15
ISDN	*Integrated services digital network*, öffentliche Kommunikationsinfrastruktur für vielfältige Dienste mit digitaler Übertragungstechnik.	11
ISO	*International Organization for Standardization*, Dachverband der nationalen Normungsinstitute.	6
ITU	*International Telecommunication Union*, internationaler Dachverband der nationalen Netzbetreiber-Gesellschaften.	5
IUT	*Implementation under test*, zu testende Implementierung beim Konformitätstest oder bei der Spuranalyse.	11
LARGE	*Learning arithmetical rules from good examples*, neuer Algorithmus zum Lernen arithmetischer Klassifikationsregeln aus positiven Beispielen.	83
LEDA	*Library of efficient data types and algorithms*, C++-Bibliothek mit zahlreichen Standard-Datentypen.	115
LOTOS	*Language for the temporal ordering specification of observational behaviour*, standardisierte prozeßalgebra-basierte Spezifikationssprache.	6
LTS	*Labelled transition system*, Kalkül zur semantischen Fundierung von Prozeßalgebren.	6
MDL	*Minimum description length*, Paradigma des machinellen Lernens, bei dem die Länge der Hypothese, ausgedrückt in der gewählten Hypothesensprache, minimiert werden soll.	82
NEA	*Nichtdeterministischer endlicher Automat.*	111
OSI	*Open systems interconnection*, Referenzmodell der ISO für offene Kommunikationssysteme.	6
OTEFSM	*Observable timed extended finite state machine*, formales Kalkül zur Spuranalyse und formale Grundlage von *FollowSM*.	33
PICS	*Protocol implementation and conformance statement*, standardisiertes Dokument, das die implementierungsspezifischen Eigenschaften einer Protokollimplementierung beschreibt.	21
PDU	*Protocol data unit*, atomare Dateneinheit bei der protokollbasierten Kommunikation.	10
SAP	*Service access point*, Dienstzugangspunkt, Schnittstelle für den Zugriff eines Benutzers auf eine Protokollinstanz.	10
SDL	*Specification and description language*, standardisierte automatenbasierte Spezifikationssprache.	6

Abkürzung	Bedeutung	Seite
SDL/GR	*SDL graphical representation*, Diagrammdarstellung von SDL-Spezifikationen.	9
SDL/PR	*SDL printed representation*, Textdarstellung von SDL-Spezifikationen.	9
SSCOP	*Service specific connection-oriented procedure*, Schicht-2-Protokoll der UNI-Signalisierung im Breitband-ISDN.	11
TCP	*Transmission control protocol*, Transportprotokoll des Internet.	117
TTCN	*Tree and tabular combined notation*, standardisierte Beschreibungssprache für Konformitätstests.	11
UNI	*User-network interface*, Teilnehmer-Netz-Schnittstelle des Breitband-ISDN.	69
UT	*Upper tester*, Testkomponente, die die Dienstschnittstelle der IUT bedient und überwacht.	12
VPCI	*Virtual path connection identifier*, Bezeichner einer virtuellen Pfadverbindung im ATM-Netz.	75

Anhang C

Literatur

[A+88] A. V. Aho et al. An optimization technique for protocol conformance test generation based on UIO sequences and rural chinese postman tours. In *Protocol Specification, Testing, and Verification, VIII* [IFI88], pages 75–86.

[BG92] B. Baumgarten and A. Giessler. *OSI-Testmethodik und TTCN.* Number 202 in Berichte der GMD. R. Oldenbourg Verlag, München, Wien, 1992.

[BM95] Ivan Bratko and Stephen Muggleton. Applications of Inductive Logic Programming. *Communications of the ACM*, 38(11):65–70, November 1995.

[Boc78] Gregor V. Bochmann. Finite state description of communication protocols. *Computer Networks*, 2:361–372, 1978.

[Bri88] Ed Brinksma. A theory for the derivation of tests. In *Protocol Specification, Testing, and Verification, VIII* [IFI88], pages 63–74.

[BvBDS91] O. B. Bellal, G. v. Bochmann, M. Dubuc, and F. Saba. Automatic test result analysis for high-level specifications. Technical report #800, University of Montreal, Department IRO, 1991.

[CCI87] CCITT. Recommendation Z.100: Specification and Description Language SDL. Contribution Com X-R15-E, CCITT, 1987.

[CCI88] CCITT. Interface between Data Terminal Equipment (DTE) and Datacircuit-Terminating Equipment (DCE) for Terminals Operating in the Packet Mode and Connected to Public Data Networks by Dedicated Circuit. Recommendation X.25, CCITT, Melbourne, 1988.

[Che76] P. P. Chen. The entity-relationship model – toward a unified view of data. *ACM Transactions on Database Systems*, 1:9–36, 1976.

[CL91] Samuel T. Chanson and Jeffrey K. H. Lo. Open systems interconnection passive monitor OSI-PM. In *Protocol Test Systems 3*, pages 423–442. University of British Columbia, 1991.

[EvB95] S. Alan Ezust and Gregor v. Bochmann. An automatic trace analysis tool generator for Estelle specifications. *Computer Communication Review*, 25(4):175–184, October 1995. Proceedings of the ACM SIGCOMM 95 Conference, Camebridge.

[FO94] Ove Færgemand and Anders Olsen. Introduction to SDL-92. *Computer Networks and ISDN Systems*, 26:1143–1167, 1994.

[FvB91] S. Fujiwara and G. v. Bochmann. Testing non-deterministic state-machines with fault coverage. In *Protocol Test Systems 4* [IFI91].

[Gol67] E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[Han95] David-Andreas Hanisch. Analyse von Zustandsübergängen in Kommunikationsprotokollen mittels Wahrscheinlichkeiten. Diplomarbeit, Technische Universität Berlin, Fachbereich Informatik, 1995.

[Hat98] Sven Hattenbach. Entwurf, Implementierung und Bewertung eines Protokollmonitors mit Wissenserwerbskomponente. Diplomarbeit, Technische Universität Berlin, Fachbereich Informatik, 1998.

[HKP91] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley Publishing Company, Redwood City, Cal., Menlo Park, Cal, et. al., 1991.

[HMS91] Peter Horn, Hubert Martens, and Werner Strobl. Zweimal messen = doppelt gut? *Datacom*, (8), 1991.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, Englewood Cliffs, N.J., London et. al., 1985.

[Hog89] Dieter Hogrefe. *Estelle, LOTOS und SDL: Standard-Spezifikationssprachen für verteilte Systeme*. Springer Compass. Springer-Verlag, Berlin, Heidelberg, New York etc., 1989.

[HS78] Ellis Horowitz and Sartaj Sahni. *Fundamentals of Computer Algorithms*. Computer Science Press, 1978.

[HT97] Lex Heerink and Jan Tretmans. Refusal testing for classes of transition systems with inputs and outputs. In *FORTE/PSTV'97 International Conference on Formal Description Techniques* [IFI97].

[IFI88] IFIP. *Proceedings of the IFIP WG 6.1 8th International Symposium on Protocol Specification, Testing, and Verification (1988)*, Amsterdam, 1988. North-Holland.

[IFI91] IFIP. *Protocol Test Systems 4, Proceedings of the 4th International Workshop on Protocol Test Systems*, Amsterdam, 1991. Elsevier Science Publishers, North Holland.

[IFI97] IFIP. *FORTE/PSTV'97 International Conference on Formal Description Techniques*. Elsevier, 1997.

[ISO84] ISO. Information Processing Systems – Open Systems Interconnection – Basic Reference Model. International Standard ISO/IS 7498, ISO, 1984.

[ISO87a] ISO. ESTELLE: A formal description technique based on an extended state transition model. International Standard ISO/IS 9074, ISO, 1987.

[ISO87b] ISO. LOTOS: Language for the temporal ordering specification of observational behaviour. International Standard ISO/IS 8807, ISO, 1987.

[ISO91] ISO. OSI conformance testing methology and framework. International Standard ISO/IS-9646, ISO, 1991.

[IT94a] ITU-T. B-ISDN ATM Adaption Layer – Service Specific Connection Oriented Protocol (SSCOP). Draft new Recommendation Q.2110, ITU-T, 1994.

[IT94b] ITU-T. Digital Subscriber Signalling System No. 2 (DSS 2). User network interface (UNI) layer 3 specification for basic call/connection control. Draft new Recommendation Q.2931, ITU-T, 1994.

[JvB83] Claude Jard and Gregor v. Bochmann. An approach to testing specifications. *The Journal of Systems and Software*, 3:315–323, 1983.

[Kai72] Richard Y. Kain. *Automata Theory: Machines and Languages*. McGraw-Hill, 1972.

[KCV92] M. C. Kim, Samuel T. Chanson, and Son T. Vuong. Protocol trace analysis based on formal specifications. In K. R. Parker, editor, *Formal Description Techniques (FORTE), IV*, pages 393–408. IFIP, North-Holland, 1992.

[KMU95] Pekka Kilpeläinen, Heikki Mannila, and Esko Ukkonen. Mdl learning of unions of simple pattern languages from positive examples. In Paul Vitányi, editor, *EuroCOLT '95*, volume 904 of *LNAI*, pages 252–260, Berlin, Heidelberg, New York, March 1995. Springer-Verlag.

[L+96] Nada Lavrač et al. ILPNET repositories on WWW: Inductive Logic Programming systems, datasets and bibliography. *AI Communications*, (9):157–206, 1996.

[Lam95] Leslie Lamport. *Das LaTeX-Handbuch*. Addison-Wesly Publishing Company, Bonn; Paris; Reading, Mass. et. al., 1995.

[Law98] Lawrence Berkeley National Laboratory Network Research Group, ftp://ftp.ee.lbl.gov/tcpdump.tar.Z. *TCPDUMP 3.4*, 1998.

[LD94] Nada Lavrač and Sašo Džeroski. *Inductive Logic Programming. Techniques and Applications*. Ellis Horwood, New York, London etc., 1994.

[LL91] D. Y. Lee and J. Y. Lee. A well-defined Estelle specification for the automatic test generation. *IEEE Transactions on Computers*, 40(4):526–542, April 1991.

[Lut93] Andreas Lutsch. Ein Parser für SDL. Technischer Bericht, Humboldt-Universität zu Berlin, 1993.

[McC90] Tom McCusker. How to decode 150 protocols. *Datamation*, December 1990.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*. Number 92 in Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York etc., 1980.

[MNU] Kurt Mehlhorn, Stefan Näher, and Christian Uhrig. *The LEDA User Manual*. LEDA Software GmbH, Saarbrücken, version 3.5.1 edition. http://www.mpi-sb.mpg.de/LEDA/leda.html.

[Mol82] M. K. Molloy. Performance analysis using stochastic petri nets. *IEEE Transactions on Computers*, 31(9):913–917, 1982.

[Mus94] Marek Musial. Entwurf eines universellen passiven Protokoll-Monitors. Diplomarbeit, Technische Universität Berlin, Fachbereich Informatik, 1994.

[Mus97] Marek Musial. On-line timed protocol trace analysis based on uncertain state descriptions. In *FORTE/PSTV'97 International Conference on Formal Description Techniques* [IFI97].

[Pil97] Peter Pille. Entwurf und Implementierung eines SDL-Übersetzers als Schnittstelle zum Protokollmonitor *FuzzySM*. Master's thesis, Technische Universität Berlin, Fachbereich Informatik, November 1997.

[Pos80] Jon Postel. DoD standard Transmission Control Protocol. Rfc (request for comments) 0761, Information Sciences Institute, University of Southern California, 1980.

[Rei85] W. Reisig. *Petri Nets*. Springer-Verlag, Berlin, Heidelberg et. al., 1985.

[RM86] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing: Explorations in the Microstrucure of Cognition*, volume 1: Foundations. MIT Press, Cambridge, 1986.

[Sch94] Ina Kathrin Schieferdecker. *Performance-Oriented Specification of Communication Protocols and Verification of Deterministic Bounds of their Qos Characteristics*. PhD thesis, Technical University of Berlin, Department of Computer Science, November 1994.

[Sie94] Gerd Siegmund. *ATM – Die Technik des Breitband-ISDN*. R. v. Decker's Verlag, Heidelberg, 1994.

[SMM98] Jürgen Suppan, Dietlind Mues, and Behrooz Moayeri. Netzwerk-Analysator „Sniffer". *Datacom*, (11), 1998.

[Sol64a] R. J. Solomonoff. A formal theory on inductive inference, part i. In *Information and Control* [Sol64b], pages 1–22.

[Sol64b] R. J. Solomonoff. A formal theory on inductive inference, part ii. *Information and Control*, 7(2):224–254, June 1964.

[TKB91] J. Tretmans, P. Kars, and E. Brinksma. Protocol conformance testing: A formal perspective on ISO 9646. In *Protocol Test Systems 4* [IFI91].

[UP86] Hasan Ural and Robert L. Probert. Step-wise validation of communication protocols and services. *Computer Networks and ISDN Systems*, 11(3):183–202, March 1986.

[UW81] Siegfried Unger and Fritz Wysotzki. *Lernfähige Klassifizierungssysteme.* Akademie-Verlag, Berlin, 1981.

[vBP94] Gregor v. Bochmann and Alexandre Petrenko. Protocol testing: Review of methods and relevance for software testing. In *Proceedings of the 1994 International Symposium on Software Testing and Analysis (ISSTA)*, ACM SIGSOFT Software Engineering Notes, Special issue, pages 109–124, August 1994.

[vE+93] Peter van Eijk et al. The Term Processor Kimwitu. Manual and Cookbook. Technical report, University of Twente, Enschede, The Netherlands, 1993.

[Zei96] E. Zeidler, editor. *Teubner-Taschenbuch der Mathematik.* B. G. Teubner, Stuttgart, Leipzig, 1996.